RooFit Introductory Tutorial

Wouter Verkerke (UC Santa Barbara)
David Kirkby (UC Irvine)

Purpose

Model the distribution of observables $\overset{®}{\mathbf{x}}$ in terms of

- Physical parameters of interest $\overset{®}{\mathbf{p}}$
- Other parameters $\overset{®}{\mathbf{q}}$ to describe detector effects (resolution,efficiency,...)

RooFit

Probability density function **F(**$\overset{®}{\mathbf{x}}$**;**$\overset{®}{\mathbf{p}}$**,**$\overset{®}{\mathbf{q}}$**)**

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

Implementation

- Add-on package to ROOT
  - ROOT is an object-oriented analysis environment
  - C++ command line interface & macros
  - Graphics interface
  - I/O support ('persisted objects')

- RooFit is collection of classes that augment the ROOT environment
  - Object-oriented data modeling
  - Integration in existing analysis environment
    - Interfaces with existing data formats
    - No need to learn new language

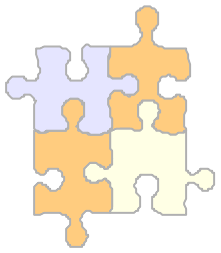
RooFit @ BaBar

- Successor of **RooFitTools**
  - **RooFitTools** no longer maintained
  - RooFit is a nearly complete rewrite (~95%) of RooFitTools
    - Class structure redesigned from scratch, having learned from RooFitTools evolution
    - Key class names and functionality identical to enhance macro portability

- Code split in two SRT packages
  - **RooFitCore**
    - Core code, base classes, interface to MINUIT, plotting logic, integrators, PDF operator classes, …
    - Everything except the PDFs
    - Maintained exclusively by Wouter & David for code stability and design overview
  - **RooFitModels**
    - PDF implementations (Gauss, Argus etc)
    - Contributed by BaBar users

- No code dependence on other BaBar software
  - Uses **SoftRelTools** for BaBar builds, but standalone Makefile provided
    - Some work still in progress…
  - Compiles clean & tested on Linux, Solaris, OSF
  - You can run it on your laptop, at home,…

The basics



Probability density functions & likelihoods

The basics of OO data modeling

The essential ingredients: PDFS, datasets, functions

Probability density functions

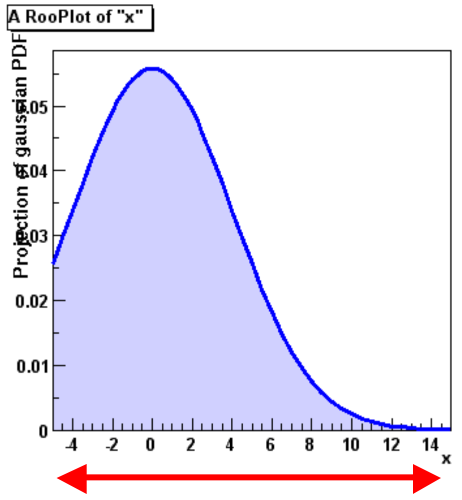
- Fundamental property of any probability density function g(x,p):

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

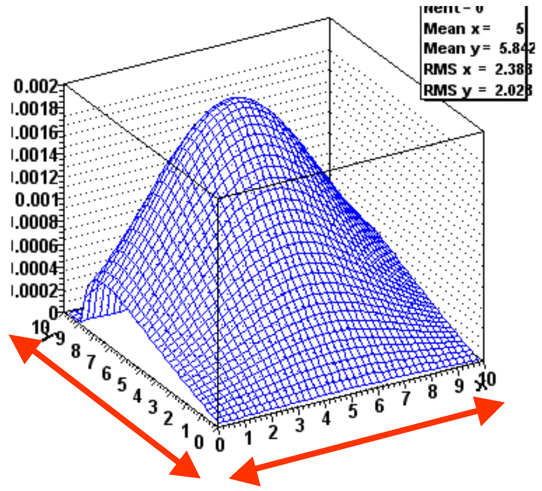
  – Easy to construct for 1-dim. PDF much more effort for >1 dim.

  – **RooFit automatically takes care of this**

    • User supplied function need not be normalized



$\int G\,dx(\vec{p}) = 1$



$\int G\,dx\,dy(\vec{p}) = 1$

Likelihood fits & ToyMC generation

- Likelihood fit
  - Likelihood is product of probabilities given by ***g(x)*** for all data points in a given dataset ***D[x]***

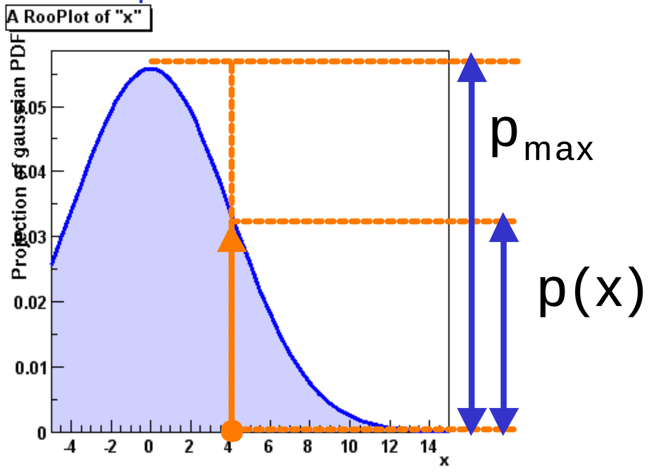
$$L(\vec{p}) = \prod_D g(\vec{x}_i, \vec{p})$$

  - Fit find $\vec{p}$ for which $-\log(L(\vec{p}))$ is smallest

$$-\log\big(L(\vec{p})\big) = -\sum_D \log(g(\vec{x}_i, \vec{p}))$$

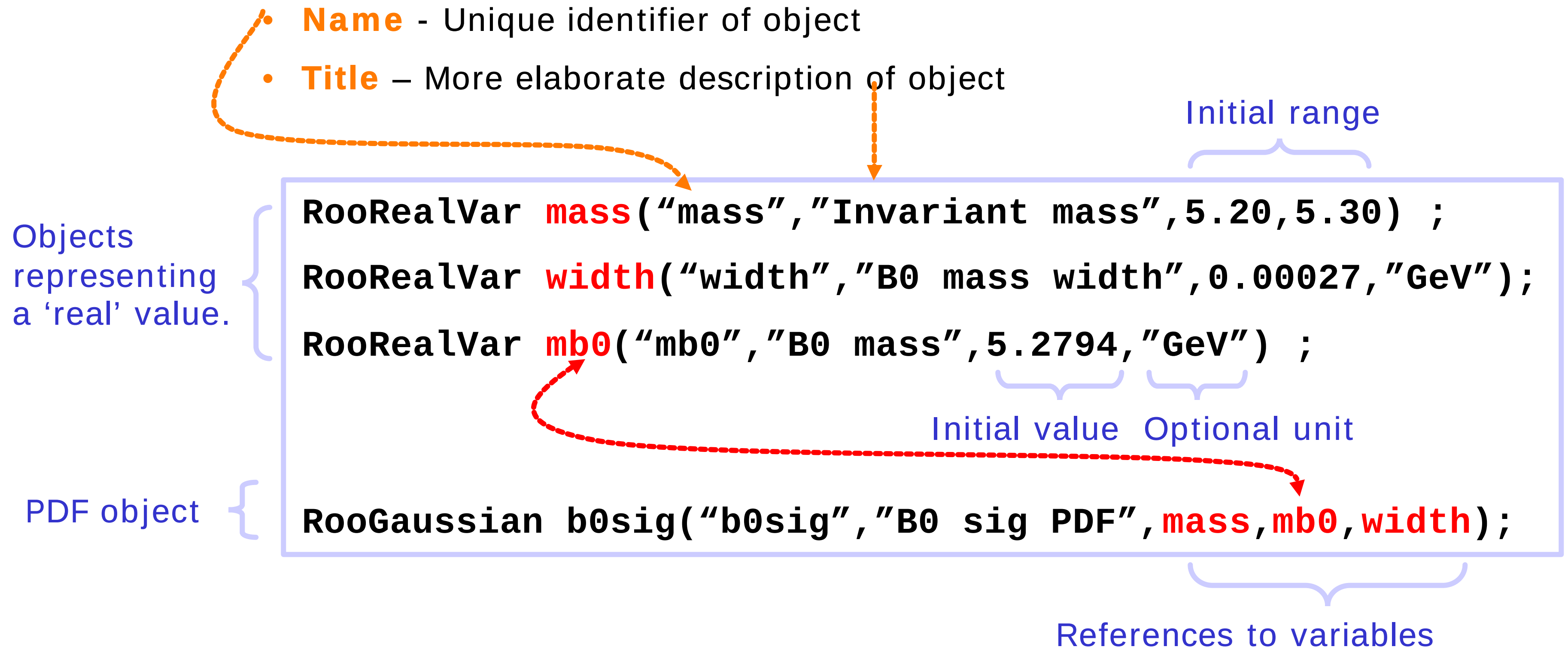
- ToyMC generation
  - Accept/reject method
  - 'Direct' method (e.g. gauss)

$P_{accept}(x) = p(x)/p_{max}$



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



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

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

Object-oriented data modeling

- Elementary operations on value holder objects

Print value and attributes

```
mass.Print()
RooRealVar::mass:  5.2500 L(5.2 - 5.3)
```

Assign new value

```
mass = 5.27 ;
mass.setVal(5.27) ;
mass = 9.0  ;
RooAbsRealLValue::inFitRange(mass):
    value 9 rounded down to max limit 5.3
```

Error: new value out of allowed range

Retrieve contents

```
Double_t massVal = mass.getVal();
```

Print works for all RooFit objects

```
b0sig.Print()
RooGaussian::b0sig(mass,mb0,width) = 0
```

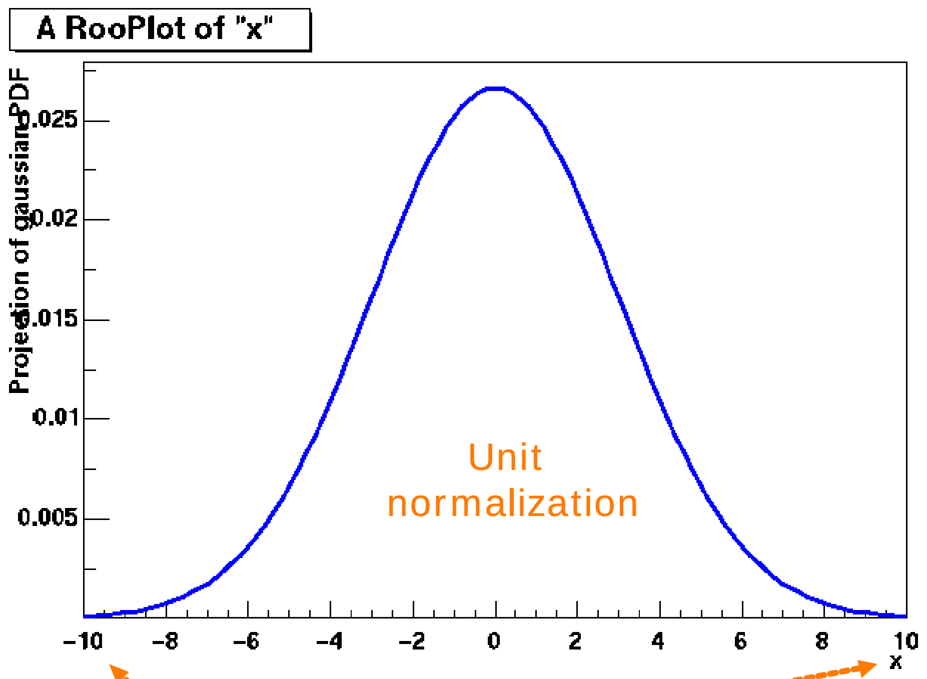
getVal() works for all real-valued objects (variables and functions)

```
Double_t val = b0sig.getVal()
```

Elementary operations with a PDF

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

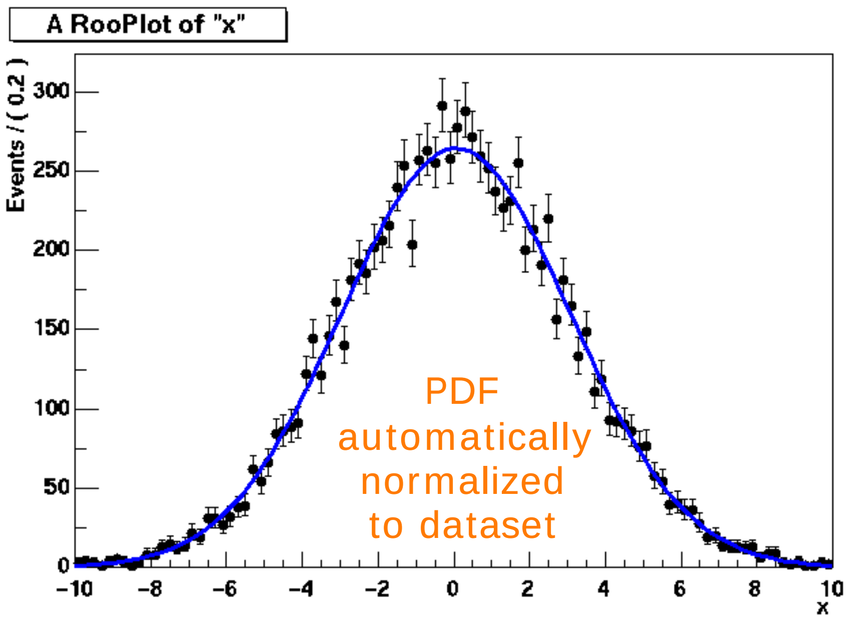
Plot range taken from limits of **x**

Elementary operations with a PDF

demo1.cc

Correct axis label for data

1) Generate 10K events from PDF
2) Fit PDF to event sample
3) Plot PDF on data



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


// Fit pdf to toy
gauss.fitTo(*data) ;



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

Once the model is built, Generating ToyMC, fitting, plotting are mostly one-line operations!

Variables → Parameter or observable?

- PDF objects have no intrinsic notion of a variable begin a parameter or observable

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

- But, PDF normalization depends on parameter/observable interpretation of variables

$$\int_{x_{\min}}^{x_{\max}} g(x,p)\,dx \equiv 1 \qquad \begin{aligned} x &= \textit{observable} \\ p &= \textit{parameter} \end{aligned}$$

- Parameter/observable interpretation is automatic and implicit when a PDF is used together with a dataset
  - All PDF variables that are member of the dataset are observables
  - All other PDF variables are parameters
  - Limits are normalization range if variable is observable
    Limits are MINUIT bounds if variable is parameter

Variables → Parameter or observable?

- Example of dynamic variable interpretation
  - BMixingPDF(dt,mixState,…) + data(dt)
    - mixState is parameter.
    - Data is fitted with pure mixed or unmixed PDF depending on value of mixState
  - BMixingPDF(dt,mixState,…) + data(dt,mixState)
    - mixState is observable.
    - PDF is normalized explicitly over the 2 states of mixState and behaves like a 2-dimensional PDF

- Determining the parameters/observables of a given PDF

getDependents:
Make list of common variables between data and gauss

```
RooArgSet* paramSet = gauss.getDependents(data) ;
paramSet.Print("v") ;
 RooArgSet::dependents:
  1) RooRealVar::x :  0 L(-10 - 10)
```

getParameters:
Make list of variables of gauss that do *not* occur in data

```
RooArgSet* paramSet = gauss.getParameters(data) ;
paramSet.Print("v") ;
RooArgSet::parameters:
  1) RooRealVar::mean  : -0.940910 +/- 0.0304
  2) RooRealVar::sigma :  3.0158 +/- 0.0222
```

Lists and sets

- RooFit has two collection classes that are frequently passed as arguments or returned as argument

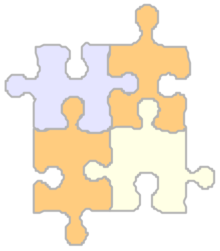
- `RooArgSet` – Set semantics
  - Each element may appear only once
  - No ordering of elements

```
RooArgSet s1(x,y,z) ;
RooArgSet s2(x,x,y) ; //ERROR!
```

- `RooArgList` – List semantics
  - Elements may be inserted multiple times
  - Insertion order is preserved

```
RooArgList l1(z,y,x) ;
RooArgList l2(x,x,y) ;
l2.Print() ;
RooArgList:::
  1) RooRealVar::x: "x"
  2) RooRealVar::x: "x"
  3) RooRealVar::y: "y"
```

Building PDFs



Basic PDFs

Combining building blocks via addition,multiplication

Generic real-valued functions

Plug-and-play parameters

The building blocks

- RooFitModels provides a collection of ‘building block’ PDFs

RooArgusBG	- Argus background shape
RooBCPEffDecay	– B0 decay with CP violation
RooBMixDecay	- B0 decay with mixing
RooBifurGauss	- Bifurcated Gaussian
RooBreitWigner	– Breit-Wigner shape
RooCBShape	- Crystal Ball function
RooChebychev	- Chebychev polynomial
RooDecay	- Simple decay function
RooDircPdf	- DIRC resolution description
RooDstD0BG	- D* background description
RooExponential	– Exponential function
RooGaussian	- Gaussian function
RooKeysPdf	- Non-parametric data description
Roo2DKeysPdf	- Non-parametric data description
RooPolynomial	- Generic polynomial PDF
RooVoigtian	- Breit-Wigner (X) Gaussian

- More will PDFs will follow
  - Easy to for users to write/contribute new PDFs

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**

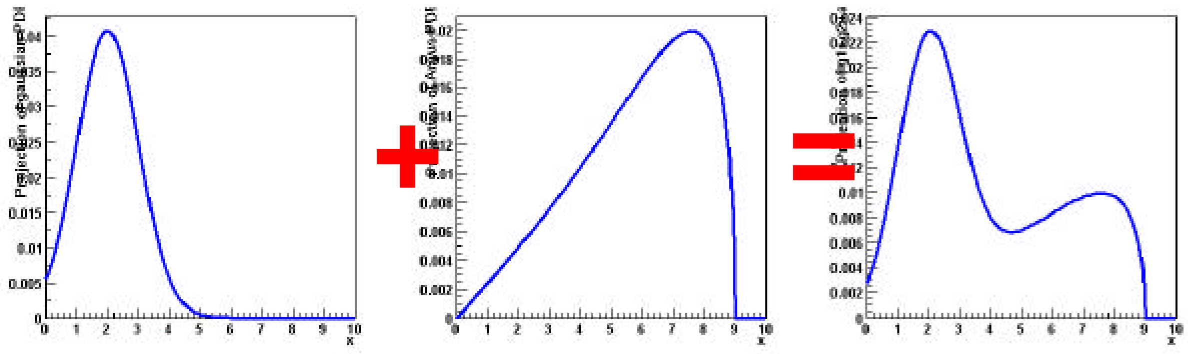
- Just write down the PDFs expression as a C++ formula

```
// 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)) ;
```

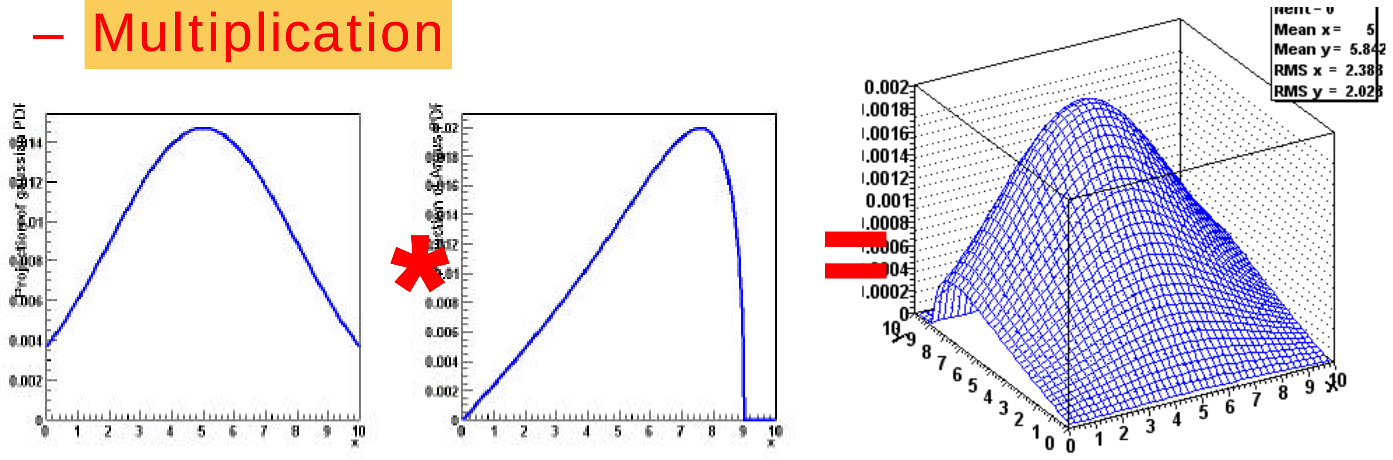
- Automatic normalization
  - Expression divided by numerical integral of expression

Building realistic models

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

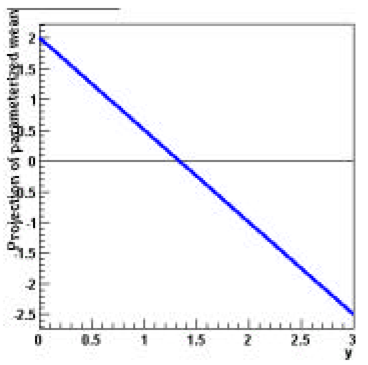


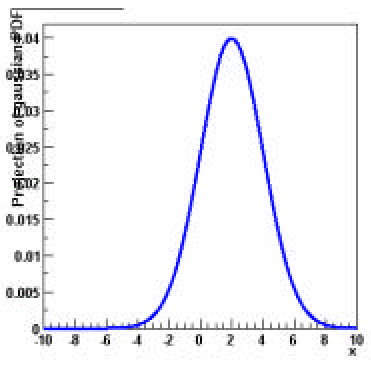
  - Multiplication



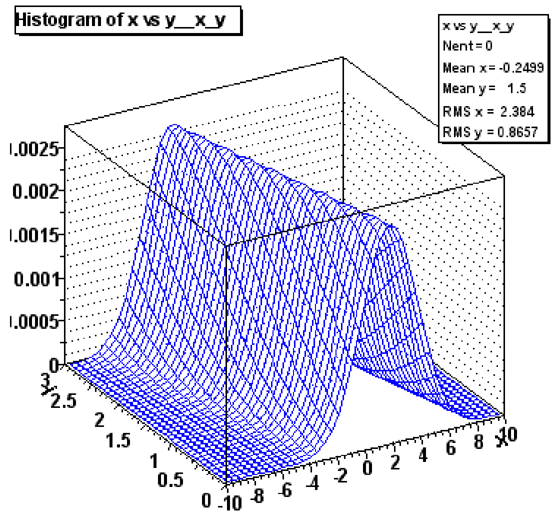
Building realistic models

– Composition ('plug & play')





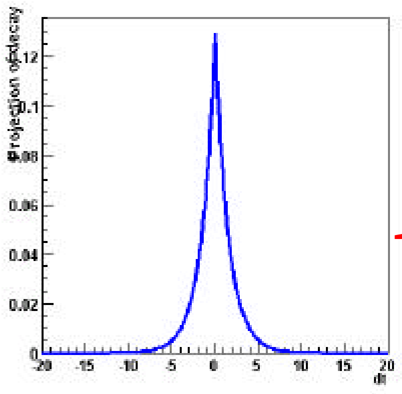
=



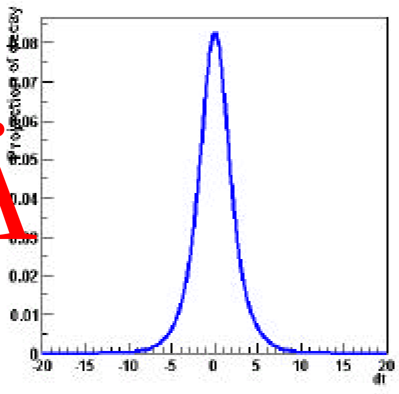
$m(y;a_0,a_1)$ $\quad$ $g(x;m,s)$ $\quad$ $g(x,y;a0,a1,s)$

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

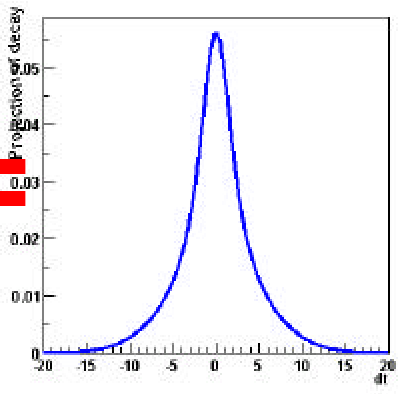
– Convolution



Ä



=



Adding PDF components

demo2.cc

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

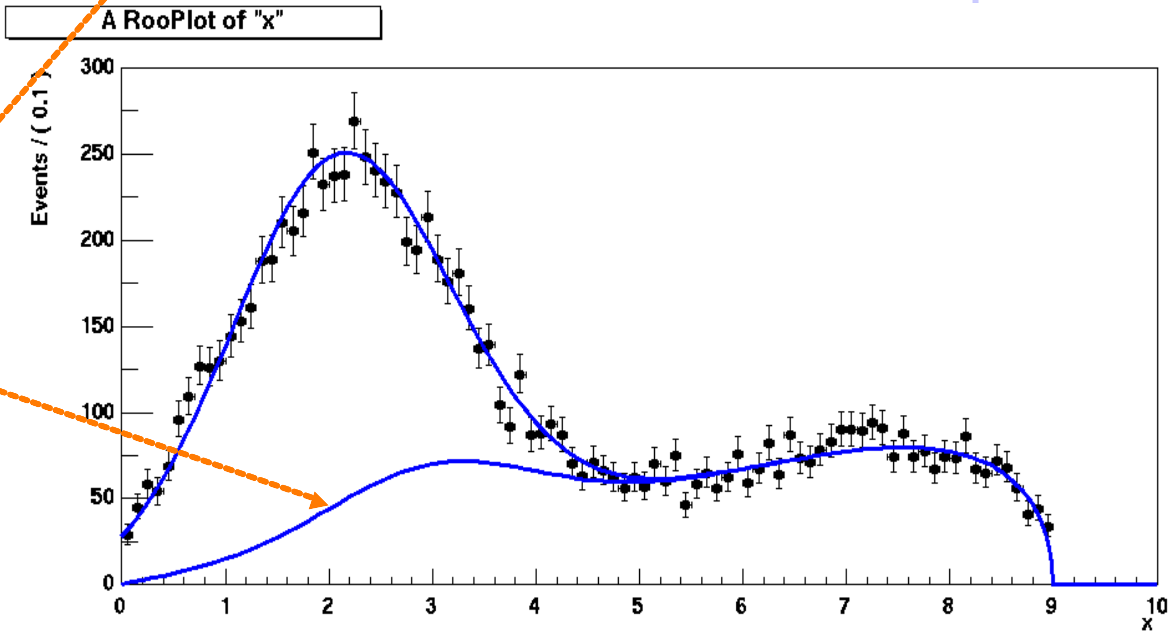
```
// Build two Gaussian PDFs
RooRealVar x("x","x",0,10) ;
RooRealVar mean1("mean1","mean of gaussian 1",2) ;
RooRealVar mean2("mean2","mean of gaussian 2",3) ;
RooRealVar sigma("sigma","width of gaussians",1) ;
RooGaussian gauss1("gauss1","gaussian PDF",x,mean1,sigma) ;
RooGaussian gauss2("gauss2","gaussian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

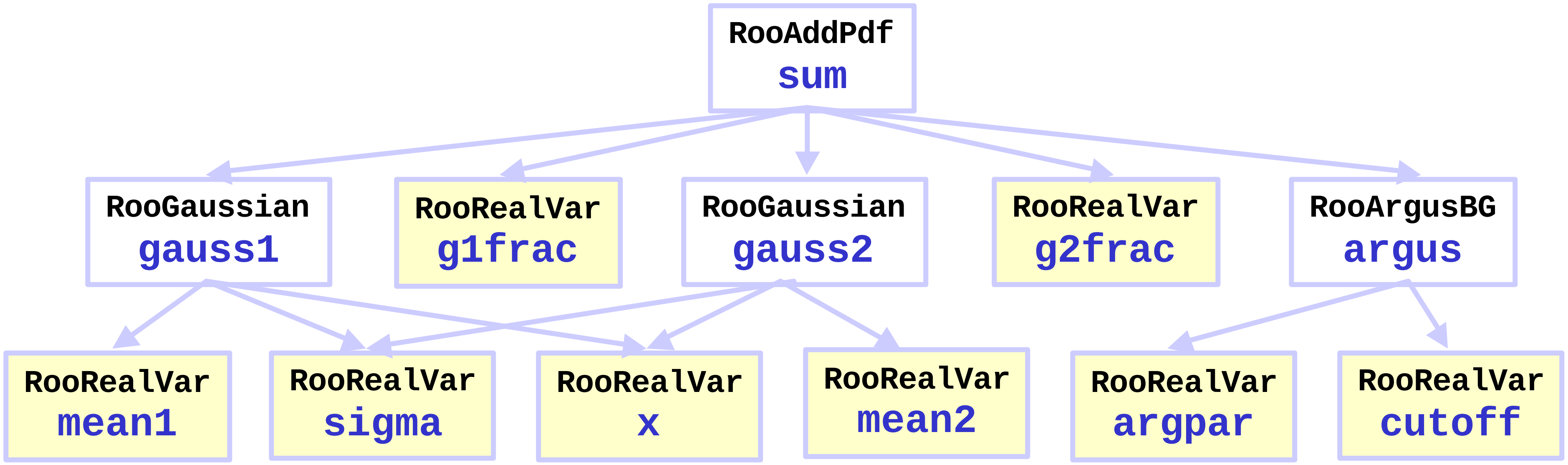
Adding PDF components

```
// 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

Parameters of composite PDF objects



```
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
```

The parameters of sum are the combined parameters of its components

Multiplying PDF components

demo3.cc

RooProdPdf constructs the product of N PDFs:

$$P = P_0(x_1, x_2) \cdot P_1(y_1, y_2,..) \cdot P_2(z_1, z_2,...) \cdot ...P_n(w_1, w_2,...)$$

Build 2 Gaussian PDFs

```
// Build two Gaussian PDFs
RooRealVar x("x","x",-5,5) ;
RooRealVar y("y","y",-5,5) ;
RooGaussian gaussx("gaussx",“gaussx",x,meanx,sigmax);
RooGaussian gaussy("gaussy",“gaussx",y,meany,sigmay);

// Multiply the components
RooProdPdf prod("gaussxy","gaussx*gaussy",
                RooArgList(gaussx,gaussy)) ;
```

Component PDFs may not share dependents
e.g. $\mathrm{pdf}_1(\mathbf{x},y)*\mathrm{pdf}_2(\mathbf{x},z)$ not allowed

Such forms are not very common,
but can be performed with RooGenericPdf
Shared parameters no problem

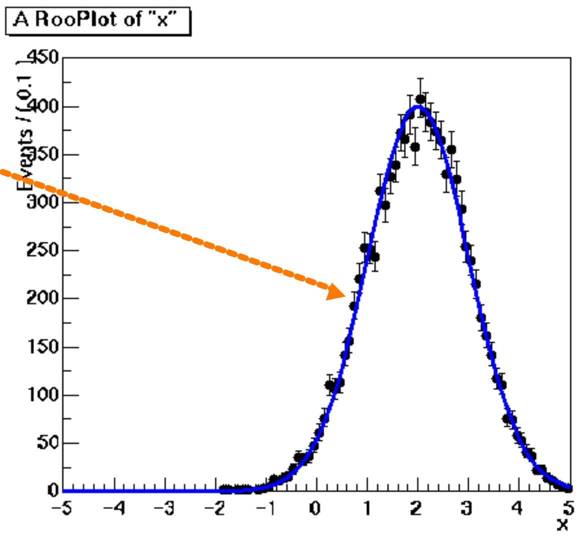
Normalization more complicated

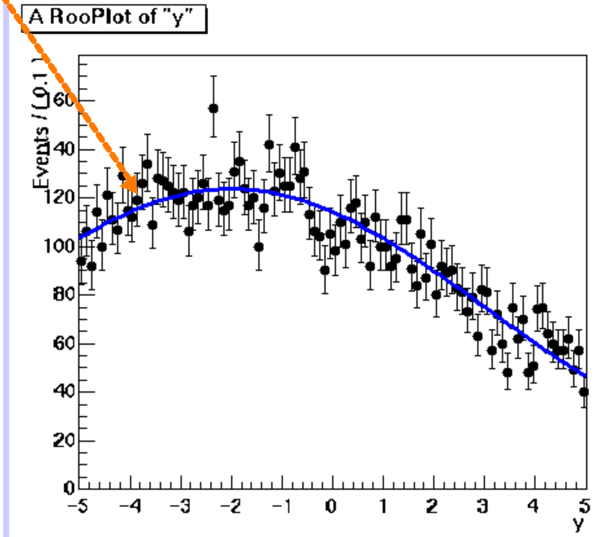
Plotting multi-dimensional PDFs

```
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

Tailoring PDFs via composition

demo4.cc

Suppose you want to build a PDF like this

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

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

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)) ;
```

Convoluted PDFs

demo6.cc

- Convoluted PDFs that can be written if the following form can be used in a very modular way in RooFit

$$P(dt,\ldots)=\sum_k c_k(\ldots)\left(f_k(dt,\ldots)\otimes R(dt,\ldots)\right)$$

coefficient

'basis function'

resolution function

Example: B^0 decay with mixing

$$c_0 = 1 \pm \Delta w, \qquad f_0 = e^{-|t|/t}$$

$$c_1 = \pm(1-2w), \qquad f_1 = e^{-|t|/t}\cos(\Delta m \cdot t)$$

Convoluted PDFs

- Physics model and resolution model are implemented separately in RooFit

Implements $f_i(dt,\ldots) \otimes R(dt,\ldots)$
Also a PDF by itself

RooResolutionModel

$$P(dt,\ldots) = \sum_k c_k(\ldots)\left(\overbrace{f_k(dt,\ldots) \otimes R(dt,\ldots)} \right)$$

$$\underbrace{c_k(\ldots)\left(f_k(dt,\ldots)\right.}$$

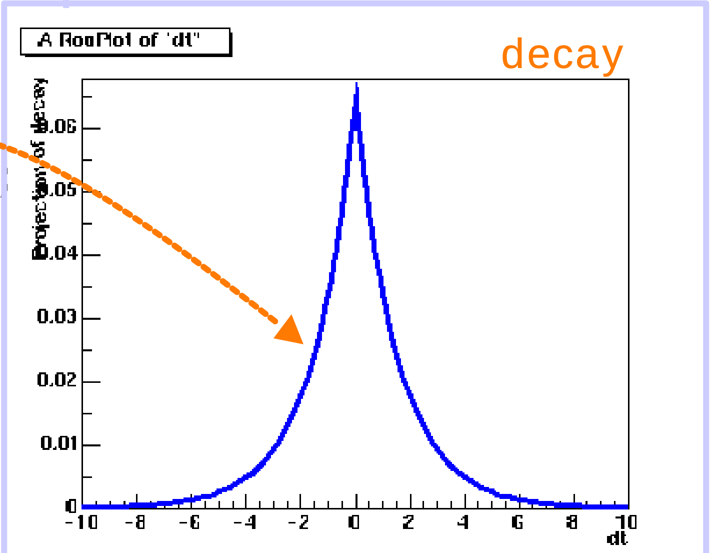
RooConvolutedPdf (physics model)

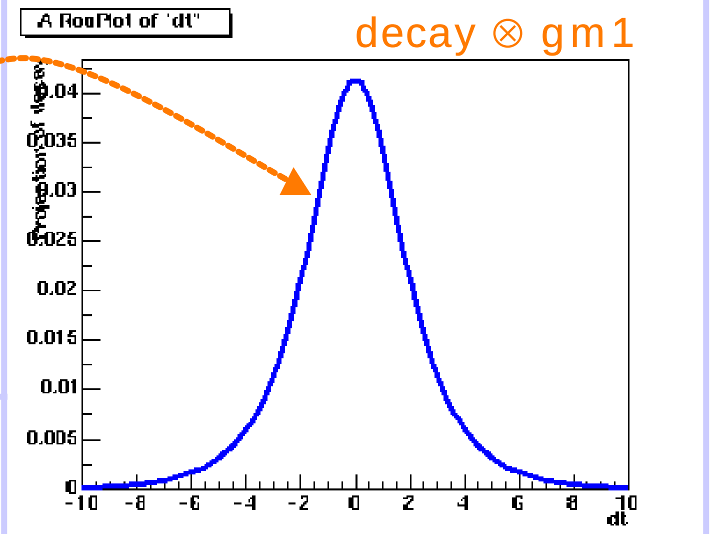
Implements $\boldsymbol{c_k}$
Declares list of $\boldsymbol{f_k}$ needed

User can choose combination of physics model and resolution model at run time
(Provided resolution model implements all f_k declared by physics model)

Convoluted PDFs

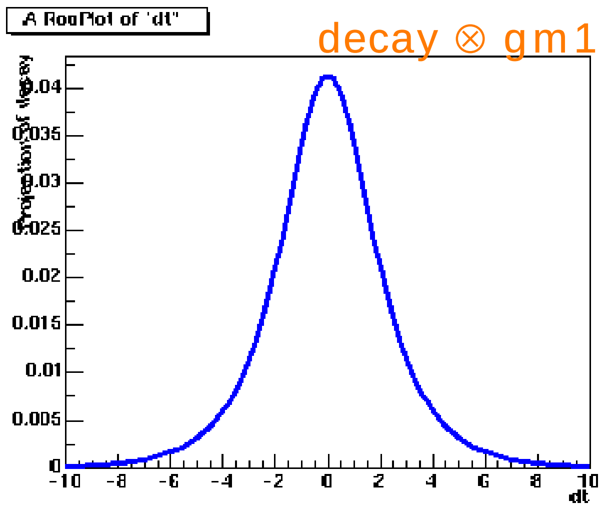
```
RooRealVar dt("dt","dt",-10,10) ;
RooRealVar tau("tau","tau",1.548) ;

// Truth resolution model
RooTruthModel tm("tm","truth model",dt) ;

// Unsmeared decay PDF
RooDecay decay_tm("decay_tm","decay",
   dt,tau,tm,RooDecay::DoubleSided) ;

// Gaussian resolution model
RooRealVar bias1("bias1","bias1",0) ;
RooRealVar sigma1("sigma1","sigma1",1) ;
RooGaussModel gm1("gm1","gauss model",
                     dt,bias1,sigma1) ;

// Construct a decay (x) gauss PDF
RooDecay decay_gm1("decay_gm1","decay",
    dt,tau,gm1,RooDecay::DoubleSided) ;
```

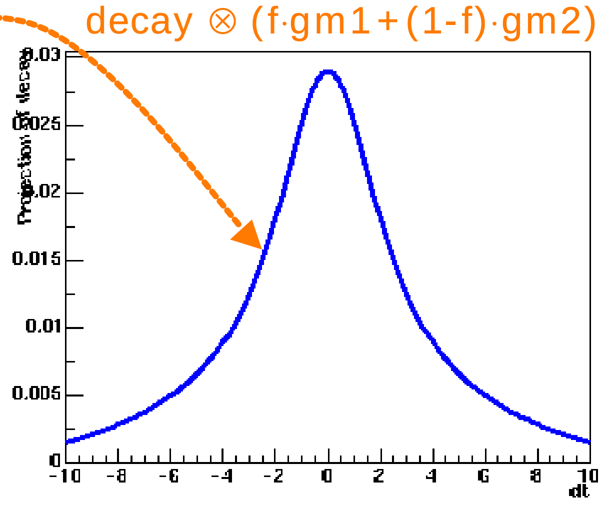




Composite Resolution Models: **RooAddModel**

```
//... (continued from last page)

// Wide gaussian resolution model
RooRealVar bias2("bias2","bias2",0) ;
RooRealVar sigma2("sigma2","sigma2",5) ;
RooGaussModel gm2("gm2","gauss model 2“
                  ,dt,bias2,sigma2) ;

// Build a composite resolution model
RooRealVar f(“f","fraction of gm1",0.5)
RooAddModel gmsum("gmsum",“gm1+gm2",
        RooArgList(gm1,gm2),f) ;

// decay (x) (gm1 + gm2)
RooDecay decay_gmsum("decay_gmsum",
              "decay",dt,tau,gmsum,
               RooDecay::DoubleSided) ;
```

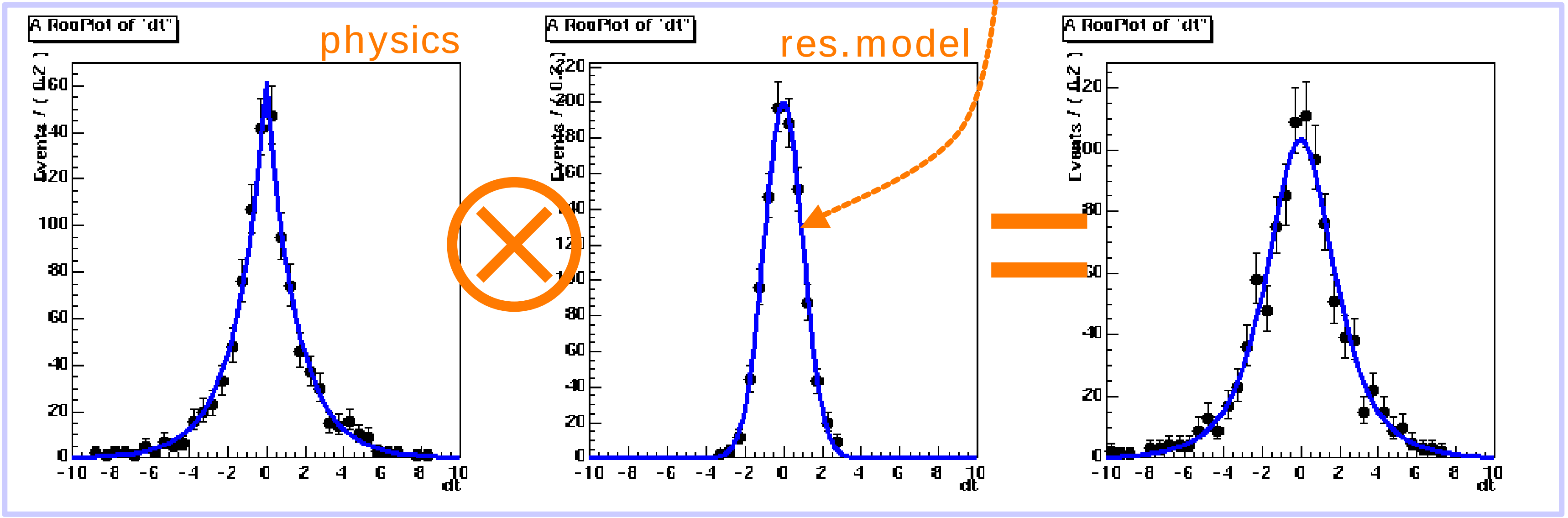




→RooAddModel works like RooAddPdf

Resolution models

- Currently available resolution models
  - **`RooGaussModel`** – Gaussian with bias and sigma
  - **`RooGExpModel`** – Gaussian (X) Exp with sigma and lifetime
  - **`RooTruthModel`** – Delta function
- A **`RooResolutionModel`** is also a PDF
  - You can use the same resolution model you use to convolve your physics PDFs to fit to MC residuals



Extended likelihood PDFs

demo9.cc

- Extended PDFs add extra term to global likelihood

$$-\log\left(L(\vec{p})\right) = -\sum_{D} \log(g(\vec{x}_i, \vec{p})) + N_{\exp} - N_{obs}\log(N_{\exp})$$

- Building extended PDFs
  - Any PDF can be turned into an extended PDF by wrapping it in a **RooExtendPdf** object

```
RooGaussian gauss(“gauss”,”Gaussian”,x,mean,sigma);
RooRealVar nsig(“nsig”,”number of signal events”,5,0,100);

RooExtendPdf gausse(“gausse”,”Extended Gauss”,gauss,nsig);
```

nsig is now a parameter of **gausse** and represents the number of expected events

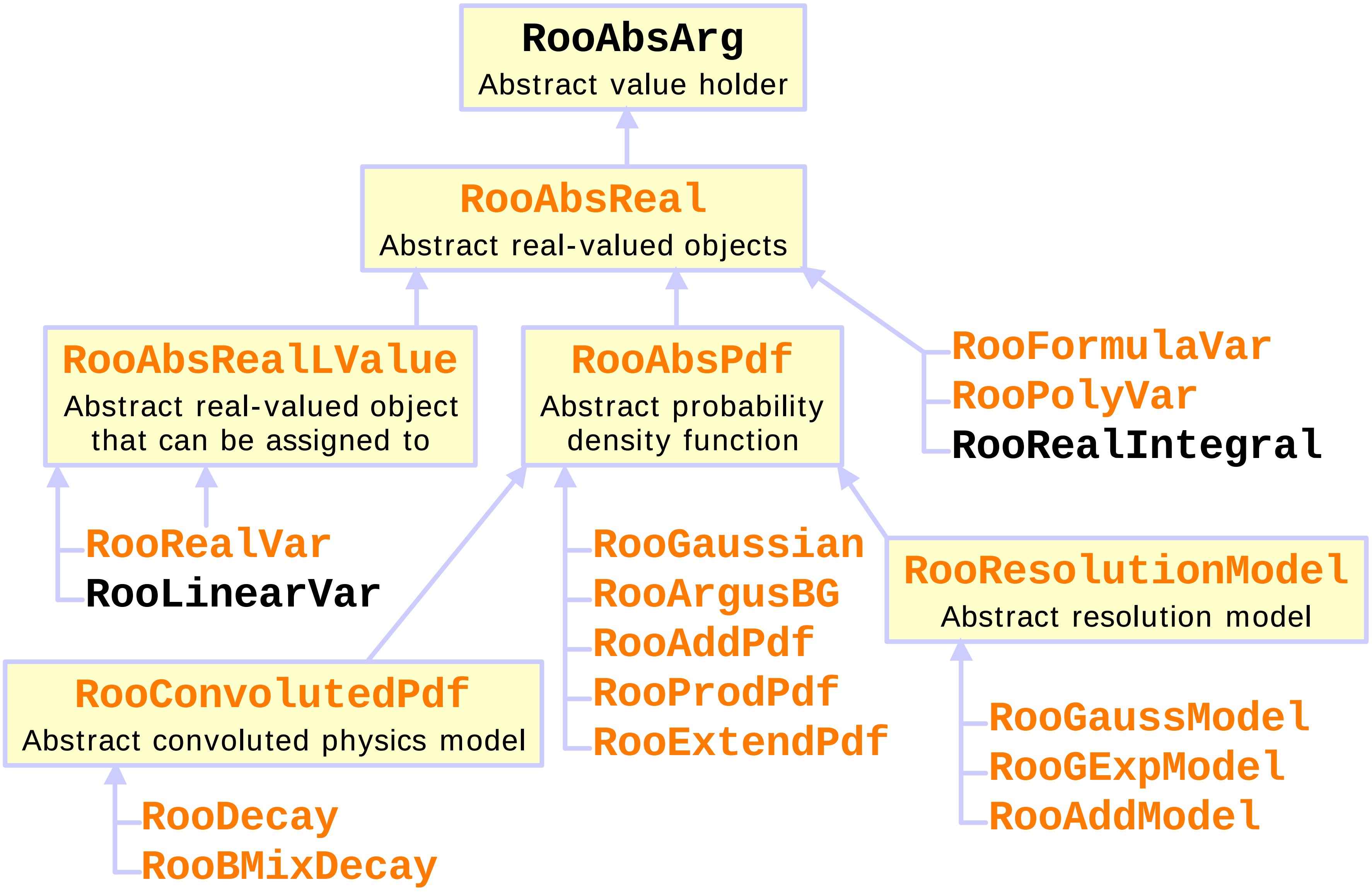
Extended likelihood PDFs

- Composition rules for extended PDFs
  - A **RooAddPdf** of all extendable PDFs is extendable
    - No coefficients needed (fractions calculated from components Nexpected)
  - A **RooProdPdf** with a single extendable component is extendable
  - A **RooSimultaneous** with any extendable component is extendable
    - Can do mixed extended/regular MLL fits in various data subsets

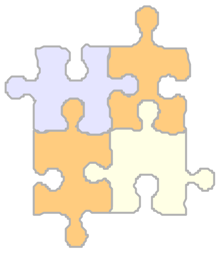
- **RooAddPdf** short-hand form for branching fraction fits
  - If **RooAddPdf** is given N coefficients instead of N-1 fractions
    $\rightarrow$ **RooAddPdf** is automatically extended
    $\rightarrow$ coefficients represent the expected #events for each PDF comp.

```
RooGaussian gauss("gauss","Gaussian",x,mean,sigma);
RooArgusBG  argus("argus","argus",x,kappa,cutoff);

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)) ;
```

Class tree for real-valued objects



Discrete variables



Organizing and classifying your data with discrete functions

Discrete-valued functions

Tabulating discrete data

Discrete variables

demo5.cc

- So far we have expressed all models purely in terms of real-valued variables
  - RooFit also has extensive support for discrete variables
  - Discrete variables are called categories
- Properties of RooFit categories
  - Finite set of named states $\rightarrow$ 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 category") ;
tagCat.defineType("Lepton") ;
tagCat.defineType("Kaon") ;
tagCat.defineType("NetTagger-1") ;
tagCat.defineType("NetTagger-2") ;
```

When to use discrete variables

- Discrete valued observables
  - B0 flavour
  - Rec/tag mixing state
- Event classification
  - tagging category
  - run block
  - B0 reconstruction mode
- Cuts
  - Mass window / sideband window
- In general, anything that
  you would use integer codes for in FORTRAN
  - RooFit makes your life easier:
    all states are labeled by name → no codes to memorize
  - Optional integer code associated with category states
    allows to import existing integer encoded data
    - Self-documenting: category state definitions provide
      single and easily understandable integer→state name conversion point

Managing data subsets / RooSimultaneous

- Simultaneous fit to multiple data samples
  - E.g. to fit PDF_A to dataset D_A and PDF_B to dataset D_B simultaneously, the NLL is

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

- Use categories to split a master dataset D into subsets D_A, D_B etc

Dataset A
5.0
3.7
1.2
4.3

Dataset B
5.0
3.7
1.2

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

Using categories: RooSimultaneous

RooSimultaneous implements 'switch' PDF:

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

$\longrightarrow$ Effectively fitting pdfA to dataA pdfB to dataB

Create dataset indexing category

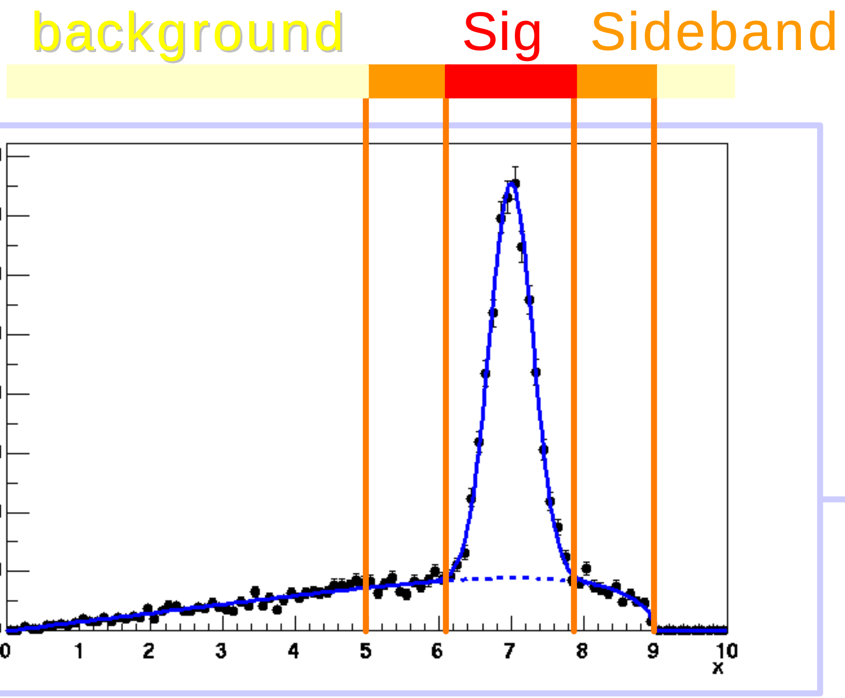
```
// Define a category with labels only
RooCategory tagCat("tagCat","Tagging category") ;
tagCat.defineType("Lepton") ;
tagCat.defineType("Kaon") ;

// Build PDFs for Lepton and Kaon data subsets

// Construct simultaneous PDF for lep and kao
RooSimultaneous simPdf("simPdf","simPdf",tagCat) ;
simPdf.addPdf(pdfLep,"Lepton") ;
simPdf.addPdf(pdfKao,"Kaon") ;
```

Associate created PDFs with appropriate index category state

Discrete functions

demo5.cc

- You can use discrete variables to describe cuts, e.g.
  - Signal, sideband mass windows
  - **RooThresholdCategory**
    - Defines regions of a real variable



```
// 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") ;

data.plotOn(someFrame,Cut("region==region::SideBand")) ;
```

Default state

Define region boundaries

Use symbolic names in future selection cuts

Wouter Verkerke, UCSB

Discrete functions

- **RooMappedCategory** provides cat $\rightarrow$ cat mapping

Define input category

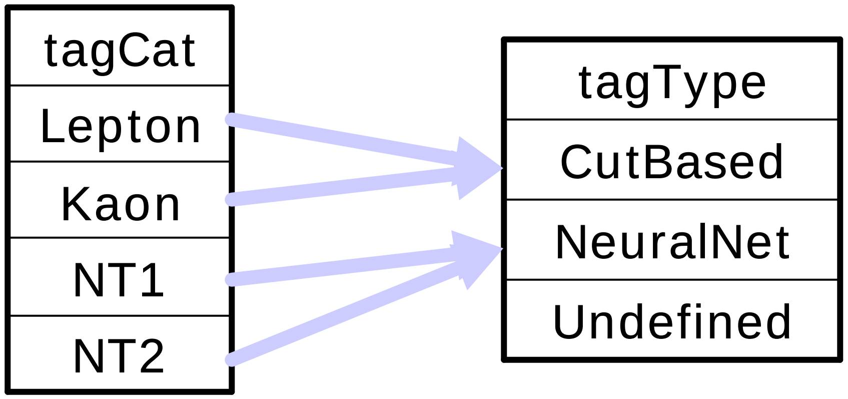
Create mapped category

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

Default state for input states without mapping rule

Wildcard expressions allowed



Discrete functions

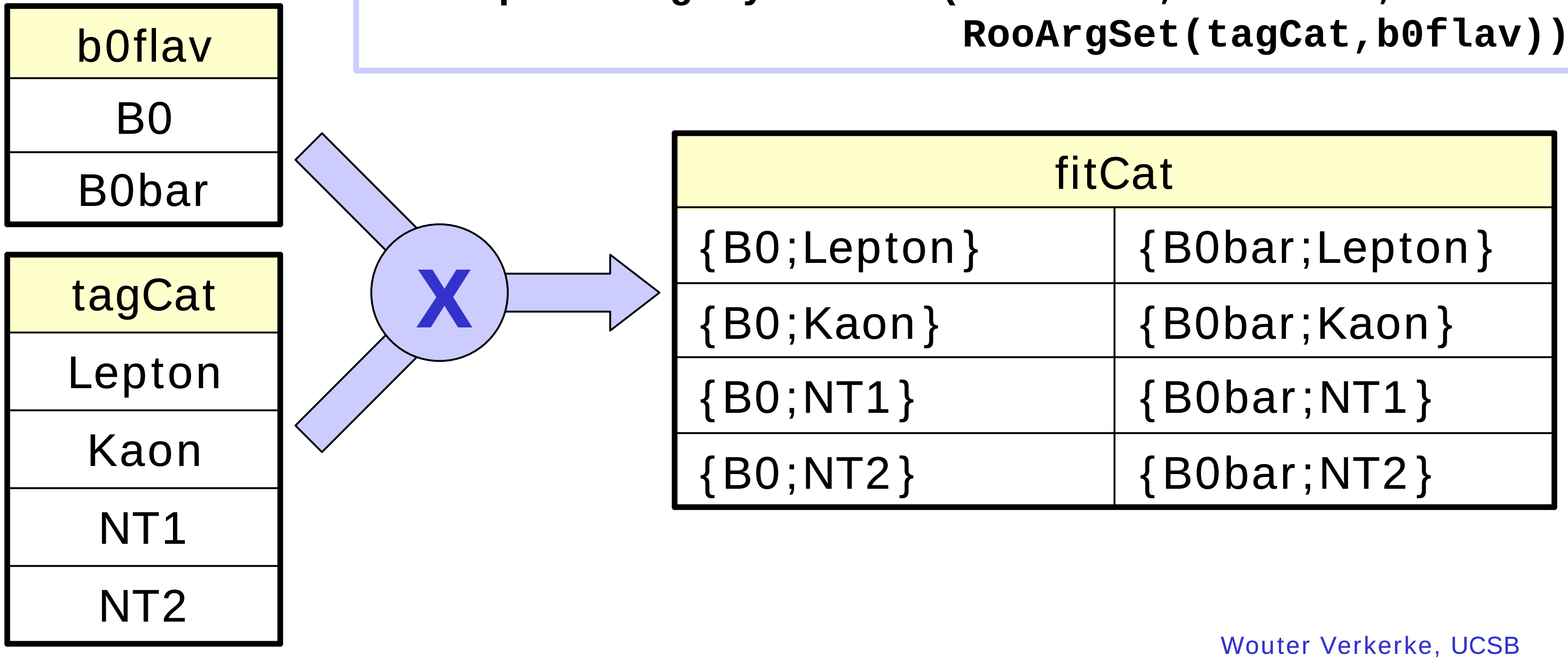
- **`RooSuperCategory/RooMultiCategory`** provides category multiplication

Define input categories

```
// Define input categories
RooCategory b0flav("b0flav","B0 flavour") ;
RooCategory tagCat("tagCat","Tagging category") ;
// state definitions omitted for clarity
```

Create super category

```
// Define ‘product’ of tagCat and runBlock
RooSuperCategory fitCat(“fitCat”,”fitCat”,
                        RooArgSet(tagCat,b0flav))
```



b0flav
B0
B0bar

tagCat
Lepton
Kaon
NT1
NT2

fitCat	
{B0;Lepton}	{B0bar;Lepton}
{B0;Kaon}	{B0bar;Kaon}
{B0;NT1}	{B0bar;NT1}
{B0;NT2}	{B0bar;NT2}

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

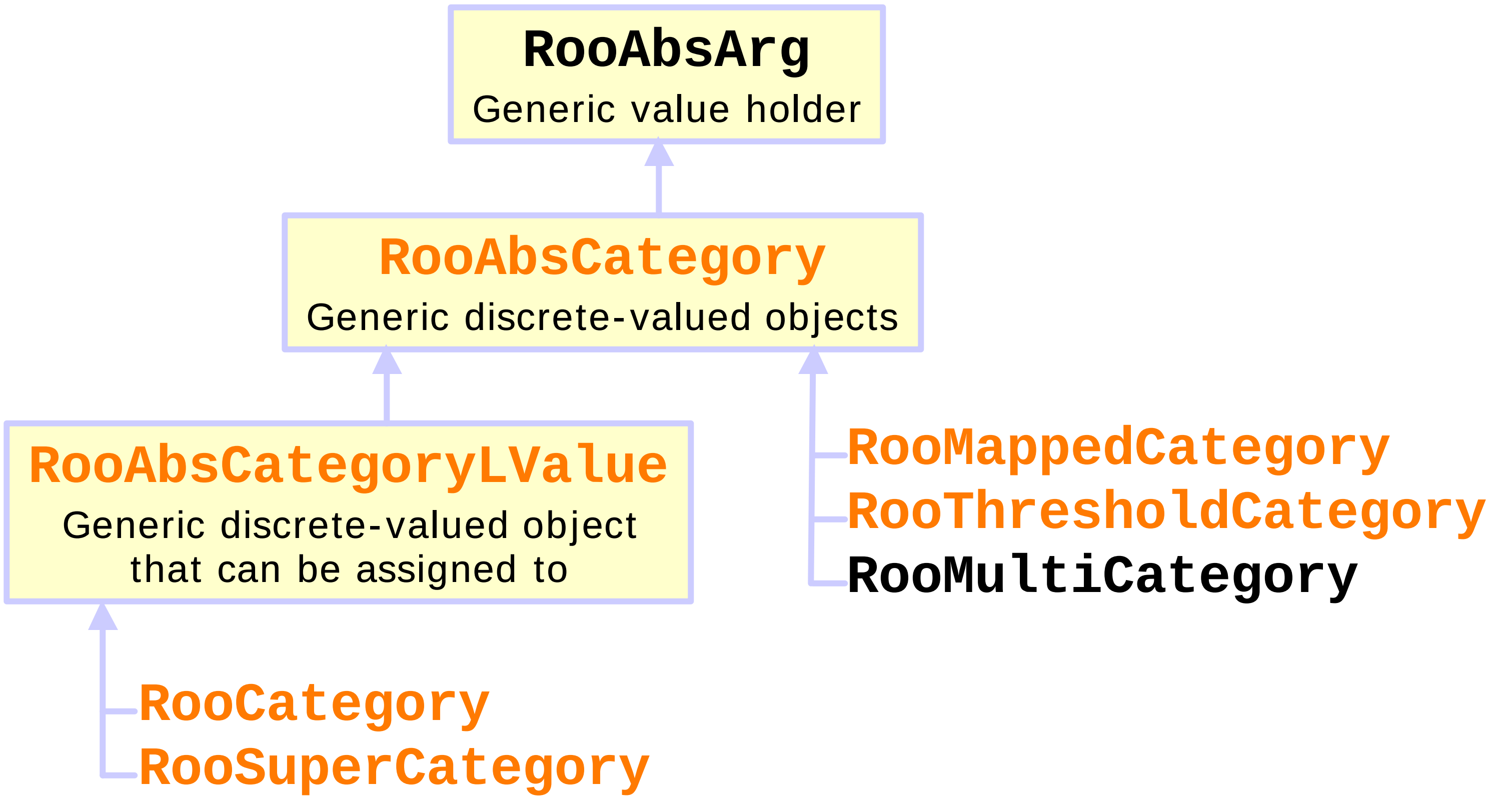
Tabulate **RooSuperCategory** states

```
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 |
  +----------------------+------+
```

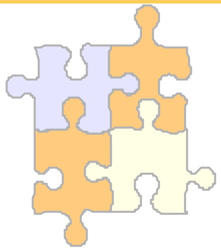
Tabulate **RooMappedCategory** states

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

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

Class tree for discrete-valued objects



Datasets



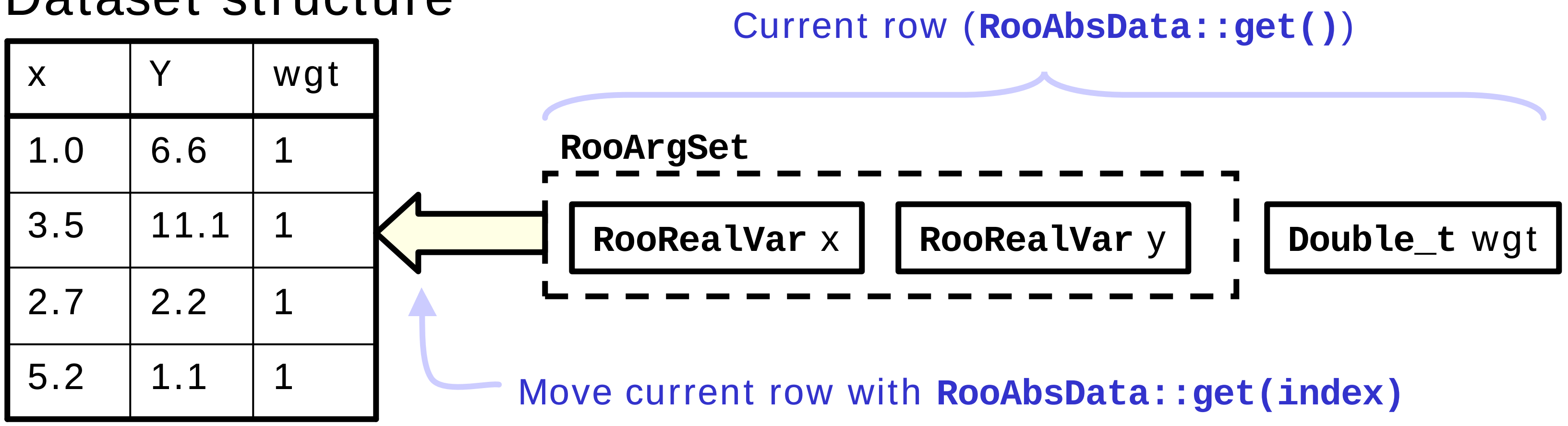
Binned vs unbinned datasets

Importing data from outside sources

Operations on datasets

An introduction to datasets

- A dataset is a collection of points in N-dimensional space
  - Dimensions can be either real or discrete
  - Two dataset implementations:
    - **RooDataSet** - unbinned (weighted & unweighted)
    - **RooDataHist** - binned
  - Common abstract base class **RooAbsData**
  - Nearly all RooFit classes/functions (*including fitting*) take **RooAbsData** objects
    - Operations universally supported for binned and unbinned data
- Dataset structure



x	Y	wgt
1.0	6.6	1
3.5	11.1	1
2.7	2.2	1
5.2	1.1	1

Current row (**RooAbsData::get()**)
RooArgSet
RooRealVar x
RooRealVar y
Double_t wgt
Move current row with **RooAbsData::get(index)**

Unbinned dataset basics

Create empty dataset with fields x,y,c. Dataset row representation will be a **clone** of (x,y,c). Original (x,y,c) will no longer be referenced after ctor.

To add a datapoint value holders x,y,c must be passed

```
// Create dataset variables
RooRealVar  x("x","x",-10,10) ;
RooRealVar  y("y","y", 0, 40) ;
RooCategory c("c","c") ;
c.defineType("Plus",+1) ;
c.defineType("Minus",-1) ;

RooDataSet
data("data","data",RooArgSet(x,y,c)) ;

// Fill d with dummy values
Int_t i ;
for (i=0 ; i<1000 ; i++) {
  x = i/50 - 10 ;
  y = sqrt(1.0*i) ;
  c = (i%2)?"Plus":"Minus" ;
  d.add(RooArgSet(x,y,c)) ;
}

d.Print("v") ;
RooDataSet::d: "d"
  Contains 1000 entries
  Defines RooArgSet::Dataset Variables:
    1) RooRealVar::x: "x"
    2) RooRealVar::y: "y"
    3) RooCategory::c: "c"
  Caches RooArgSet::Cached Variables:
```

Unbinned dataset basics

Access the pointer to the **RooArgSet** holding the current row

```
// Retrieve the 'current' row
RooArgSet* row = data.get() ;
row->Print("v") ;
RooArgSet::Dataset Variables: (Owning contents)
  1) RooRealVar::x :  9.0000 L(-10 - 10)
  2) RooRealVar::y :  31.607 L(0 - 40)
  3) RooCategory::c : Plus
```

Load row #900 in the **RooArgSet** holding the current row

```
// Retrieve a specific row
row = data.get(900) ;
row->Print("v") ;
RooArgSet::Dataset Variables: (Owning contents)
  1) RooRealVar::x :  8.0000 L(-10 - 10)
  2) RooRealVar::y :  30.000 L(0 - 40)
  3) RooCategory::c : Minus
```

Find value holder for x in the current row

```
// Retrieve a specific field of the row
RooRealVar* xrow = (RooRealVar*) row->find("x") ;
cout << xrow->getVal() << endl ;
8.0000
```

Weighting unbinned datasets

```
// Print current row and weight of dataset
row->Print(“v”) ;
RooArgSet::Dataset Variables: (Owning contents)
  1) RooRealVar::x :  8.0000 L(-10 - 10)
  2) RooRealVar::y :  30.000 L(0 - 40)
  3) RooCategory::c : Minus
cout << data.weight() << endl ;
1.0000

// Designate variable y as the event weight
data.setWeightVar(y)

// Retrieve same row again
row = data.get(900) ;
row->Print(“v”) ;
RooArgSet::Dataset Variables: (Owning contents)
  1) RooRealVar::x :  8.0000 L(-10 - 10)
  2) RooCategory::c : Minus

cout << data.weight() << endl ;
30.0000
```

Instruct dataset to interpret y as the event weight

Variable y is no longer in the current row

Current value of y is returned as the event weight

Importing data

- Unbinned datasets (**RooDataSet**) can be constructed from
  - ROOT **TTree** objects
    - RooRealVar dataset rows are taken /D /F /I tree branches with equal names
    - RooCategory dataset rows are taken from /I /b tree branches with equal names

```
Ttree* tree = <someTFile>.Get("<someTTree>") ;
RooDataSet data("data","data",tree,RooArgSet(x,c));
```

  - ASCII data data files
    - ASCII file fields are interpreted in order of supplied **RooArgList**

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

Implicit selection: External data may contain entries that exceed limits set on RooFit value holder objects

- If a loaded value of a **RooRealVar** exceeds the RRVs limits, the entire tree row is not loaded
- If a loaded index of a **RooCategory** is not defined, the entire tree row is not loaded

Importing data

- Binned dataset (**RooDataHist**) can be constructed from
  - ROOT **TH1/2/3** objects
    - TH dimensions are matched in order to supplied list of RooFit value holders

```
TH2* histo = <yourTHistogram> ;
RooDataHist bdata(“bdata”,”bdata”,RooArgList(x,y),histo);
```

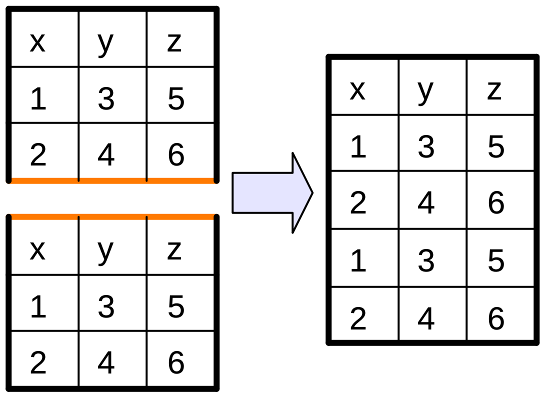
  - **RooDataSet** unbinned datasets
    - Binning for each dimension is specified by **setFitRange(lo,hi),setFitBins(nbins)**
    - The unbinned dataset may have more dimensions than the binned dataset.
      Dimensions not specified are automatically projected

```
RooDataSet* data = <yourUnbinnedData> ;
RooDataHist bdata(“bdata”,”bdata”,RooArgList(x,y),data) ;
```

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) ;
```



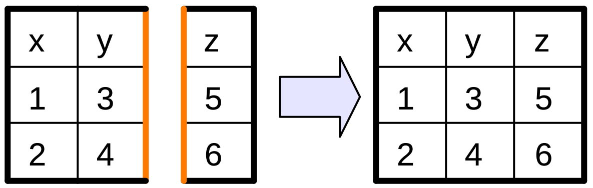
x	y	z
1	3	5
2	4	6

x	y	z
1	3	5
2	4	6

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

- Merging

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

d1.merge(d2) ;
```



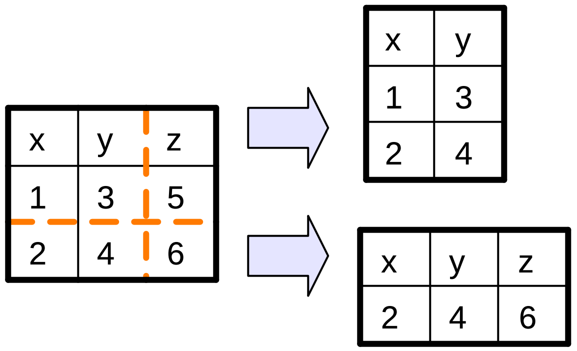
x	y
1	3
2	4

z
5
6

x	y	z
1	3	5
2	4	6

- Reducing

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

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

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



x	y	z
1	3	5
2	4	6

x	y
1	3
2	4

x	y	z
2	4	6

Adding and reducing binned datasets

- Adding

```
RooDataHist d1(“d1”,”d1”,
               RooArgSet(x,y));
RooDataHist d2(“d2”,”d2”,
               RooArgSet(x,y));

d1.add(d2) ;
```

w	y1	y2
x1	0	**1**
x2	**1**	0

w	y1	y2
x1	0	**1**
x2	**1**	0

$\Rightarrow$

w	y1	y2
x1	**1**	**1**
x2	**1**	**1**

- Reducing

```
RooDataHist d1(“d1”,”d1”,
               RooArgSet(x,y) ;

RooDataHist* d2 =
     d1.reduce(x);

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

w	y1	y2
x1	0	1
x2	1	0

$\Rightarrow$

-	w
x1	**1**
x2	**1**

w	y1	y2
x1	0	1
x2	1	0

$\Rightarrow$

w	y1	y2
x1	0	**0**
x2	1	0

Fitting & Generating

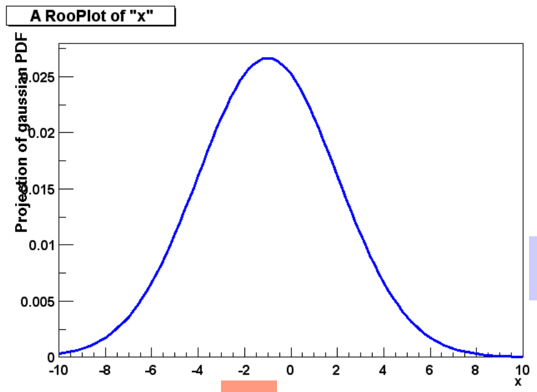
Fitting

Browsing your fit results

Generating toy MC

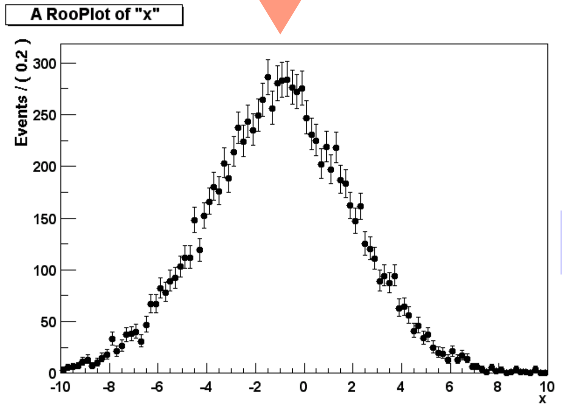
Putting it all together

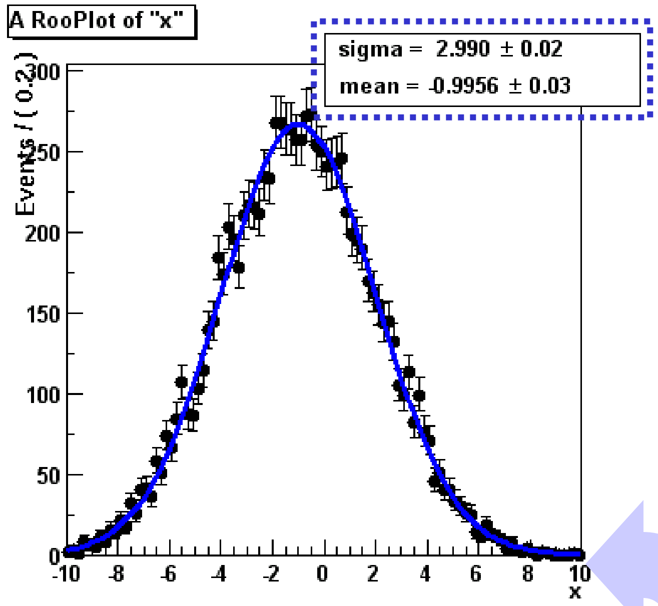
Given a model, fitting and generating are 1-line operations



Works for *any* PDF

```
data = gauss.generate(x,1000)
```





Binned or unbinned maximum likelihood fit

```
fitResult = gauss.fitTo(data)
```

Interface to MINUIT for fitting

```
 COVARIANCE MATRIX CALCULATED SUCCESSFULLY
FCN=25054.9 FROM HESSE     STATUS=OK             10 CALLS          69 TOTAL
                    EDM=3.65627e-06    STRATEGY= 1      ERROR MATRIX ACCURATE
  T PARAMETER                                    INTERNAL      INTERNAL
 NO.   NAME      VALUE            ERROR          STEP SIZE       VALUE
  1  mean      -9.95558e-01   3.01321e-02   6.59595e-04  -9.95558e-01
  2  sigma      2.99001e+00   2.20203e-02   9.66748e-05   2.99001e+00
                                 ERR DEF= 0.5
EXTERNAL ERROR MATRIX.    NDIM=  25    NPAR=  2    ERR DEF=0.5
 9.079e-04 -1.787e-05
-1.787e-05  4.849e-04
```

Fitting

```
RooAbsData* data ;
RooAbsPdf* pdf ;
RooFitResult* fitres = pdf->fitTo(*data,"<options>") ;
```

– Binned/unbinned fit performed depending on type of dataset (**RooDataHist**/**RooDataSet**)

– Fitting options:

MINUIT control options

"m" = MIGRAD only, i.e. no MINOS
"s" = estimate step size with HESSE before starting MIGRAD
"h" = run HESSE after MIGRAD
"e" = Perform extended MLL fit
"0" = Run MIGRAD with strategy MINUIT 0 (faster, but no corr. matrix at end) Does not apply to HESSE or MINOS, if run afterwards.

output options

"q" = Switch off verbose mode
"l" = Save log file with parameter values at each MINUIT step
"v" = Show changed parameters at each MINUIT step
"t" = Time fit
"r" = Save fit output in RooFitResult object (return value is object RFR pointer)

Automatic fit optimization

- RooFit analyzes PDF objects prior to fit and applies several optimizations
  - Actual fit performed on copy of PDF and dataset
    - Allows case-specific non-reversible optimizations
  - Components that have all constant parameters are pre-calculated
  - Dataset variables not used by the PDF are dropped
  - Simultaneous fits: When a parameters changes only parts of the total likelihood that depend on that parameter are recalculated
  - PDF normalization integrals are only recalculated when the ranges of their observables or the value of their parameters are changed
    - Lazy evaluation: calculation only done when intergral value is requested
- Little or no need for ‘hand-tuning’ of user PDF code
  - Easier to code and code is more readable
- ‘Typical’ large-scale fits see significant speed increase
  - Factor of 3x – 10x not uncommon.

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

Browsing fit results with **RooFitResult**

- Easy navigation of correlation matrix
  - Select single element or complete row by parameter name

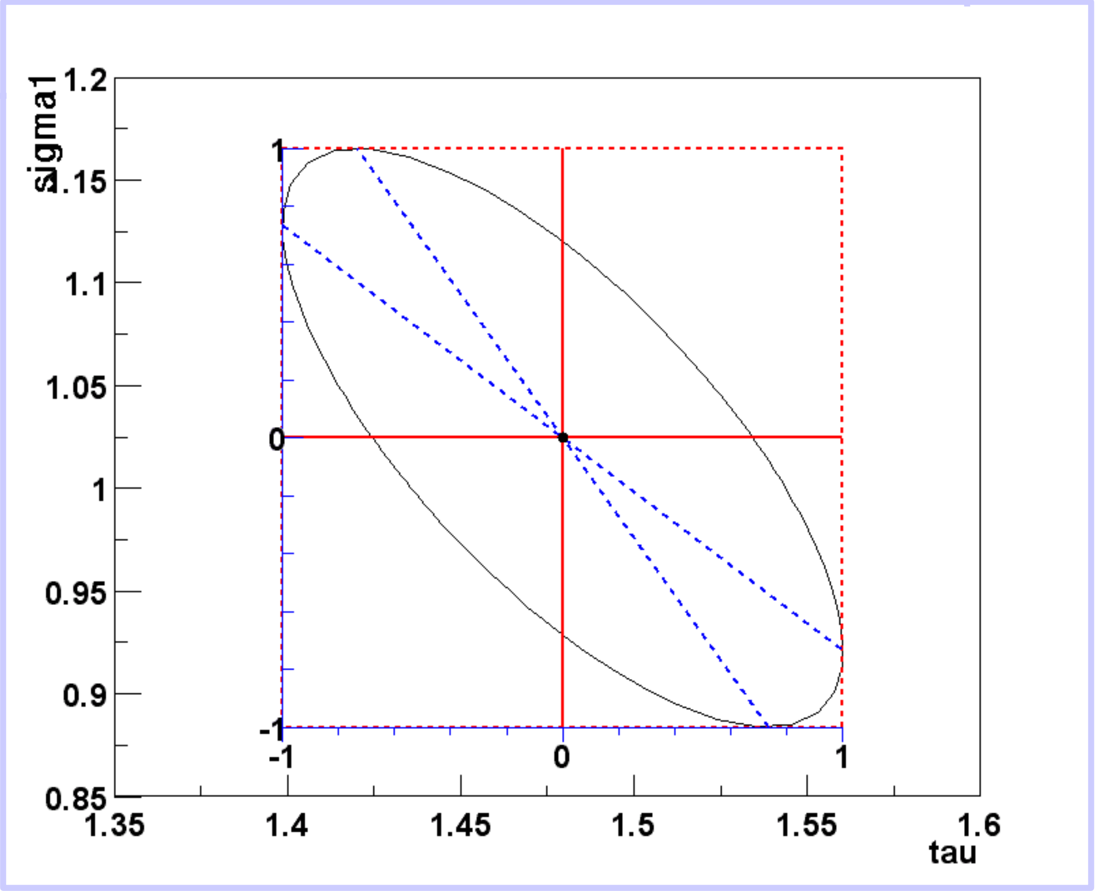
```
r->correlation("argpar","sigma")
(const Double_t)(-9.25606412005910845e-02)

r->correlation("mean1")->Print("v")
RooArgList::C[mean1,*]: (Owning contents)
  1) RooRealVar::C[mean1,argpar] :  0.11064 C
  2) RooRealVar::C[mean1,g2frac] : -0.0262487 C
  3) RooRealVar::C[mean1,mean1]  :  1.0000 C
  4) RooRealVar::C[mean1,mean2]  : -0.00632847 C
  5) RooRealVar::C[mean1,sigma]  : -0.0339814 C
```

- **RooFitResult** persistable with ROOT I/O
  - Save your batch fit results in a ROOT file and navigate your results just as easy afterwards

Visualize errors and correlation matrix elements

```
RooFitResult* r = pdf->fitTo(data,"mhvr") ;
RooPlot* f = new RooPlot(tau,sigma1,1.35,1.6,0.85,1.20) ;
r->plotOn(f,tau,sigma1,"ME12VHB") ;
f->Draw() ;
```

Works on any **RooFitResult**,
Also after persistence



MINUIT contour scan
is also possible with
a separate interface

Generating ToyMC

- Normal generator run
  - Just specify set of observables to generate and #events

```
RooAbsPdf* pdf ;
RooDataSet* toyMCdata = pdf->generate(RooArgSet(dt,mixState),10000);
```

Observables to generate

#events

- Generator run with prototype data
  - Specify set of observables to generate and a prototype dataset

```
RooDataSet* protoData
RooAbsPdf* pdf ;
RooDataSet* toyMCdata = pdf->generate(RooArgSet(dt,mixState),*protoData);
```

Observables to generate

Prototype dataset

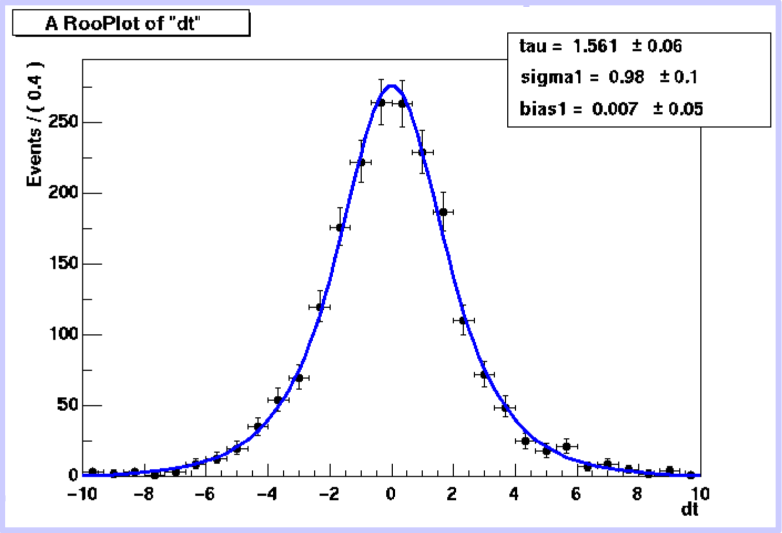
  - Generated dataset will replicate *exactly* the prototype dataset except for observables generated by the PDF
  - Ideal for per-event errors, tagging breakdown, ...

Automatic generator optimizations

- Most efficient generator technique automatically selected
  - PDF components can advertise a smarter generation technique (direct generation, e.g. gauss) which is used when appropriate
  - **`RooProdPdf`** delegates generation of observables to component PDFs (1 x N-dim generation $\rightarrow$ N x 1-dim generation)
  - **`RooAddPdf`** components generated separately
    Accept/reject method very inefficient when broad and narrow distributions are summed
  - **`RooConvolutedPdf`** generates physicsPDF and smearing model separately if both support 'direct' generation
    (convolution integrals not evaluated during generation)

Putting it all together: generating and fitting a decay PDF

```
// Build a simple decay PDF
RooRealVar dt("dt","dt",-20,20) ;
RooRealVar tau("tau","tau",1.548,-10,10) ;

// Build a gaussian resolution model
RooRealVar bias("bias","bias",0,-5,5) ;
RooRealVar sigma("sigma","sigma",1,0.1,2.0) ;
RooGaussModel gm("gm","gauss model",dt,bias,sigma) ;

// Construct a decay (x) gm
RooDecay decay(“decay","decay",dt,tau,gm,RooDecay::DoubleSided) ;

// Generate BMixing data with above set of event errors
RooDataSet *data = decay.generate(dt,2000) ;

// Fit the generated data to the model
RooFitResult* r = decay.fitTo(*data,”mhr”) ;
r->correlation(sigma,tau) ;
-0.818443

// Make a plot of the data and PDF
RooPlot* dtframe = dt.frame(-10,10,30) ;
data->plotOn(dtframe) ;
pdf.plotOn(dtframe) ;
pdf.paramOn(dtframe) ;
dtframe->Draw() ;
```



Plotting & Saving

Adding statistics, parameter boxes

Changing colors and styles

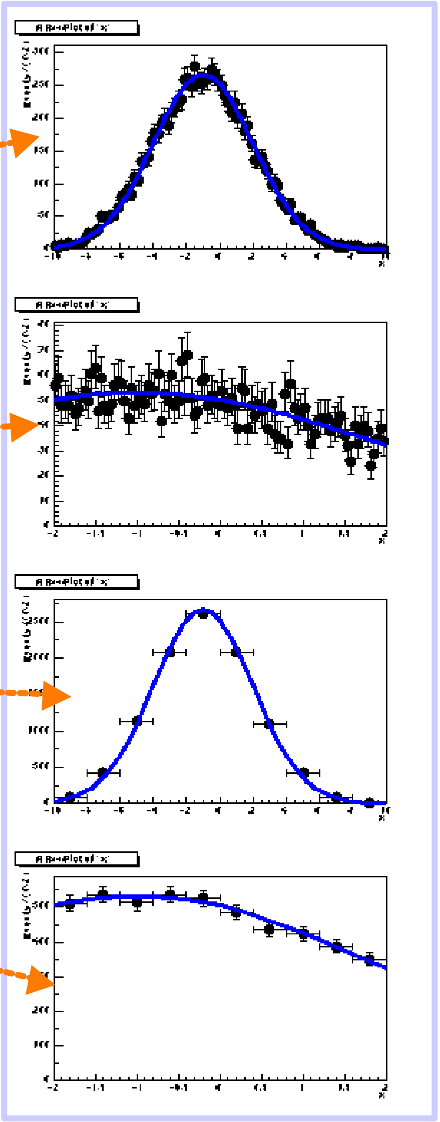
Plotting in 2 and 3 dimensions

Persisting plots & fit results

Changing the plot range / histogram binning

- By default a **RooPlot** frame takes the limits and the number of bins from its plot variable
  - Can be overridden by **frame()** arguments

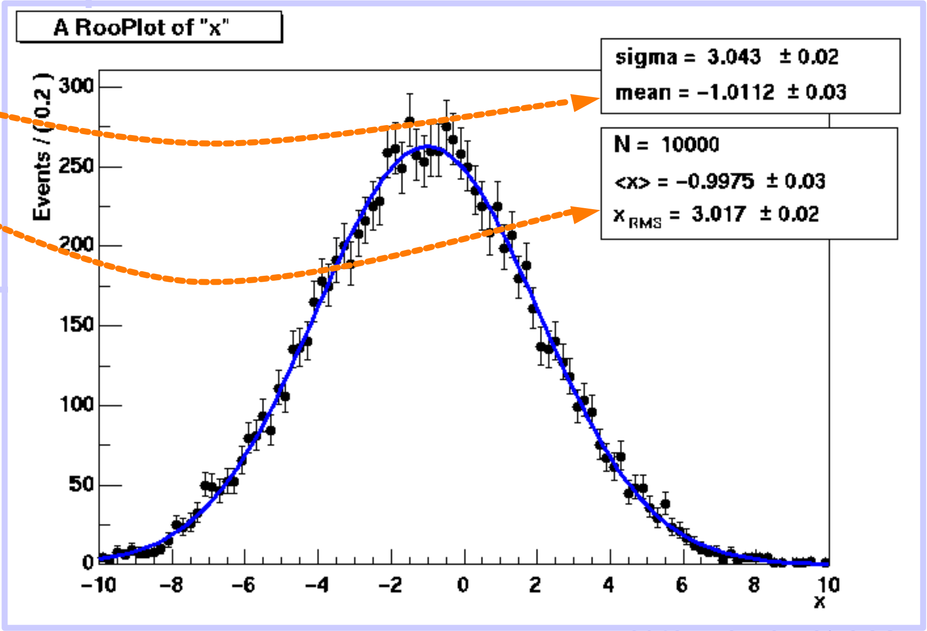
```
RooPlot* frame1 = x.frame() ;
data->plotOn(frame1) ;
pdf->plotOn(frame1) ;
frame1->Draw() ;

RooPlot* frame2 = x.frame(-2,2) ;
data->plotOn(frame1) ;
pdf->plotOn(frame1) ;
frame2->Draw() ;

RooPlot* frame3 = x.frame(10) ;
data->plotOn(frame1) ;
pdf->plotOn(frame1) ;
frame3->Draw() ;

RooPlot* frame3 = x.frame(-2,2,10);
data->plotOn(frame1) ;
pdf->plotOn(frame1) ;
frame3->Draw() ;
```



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

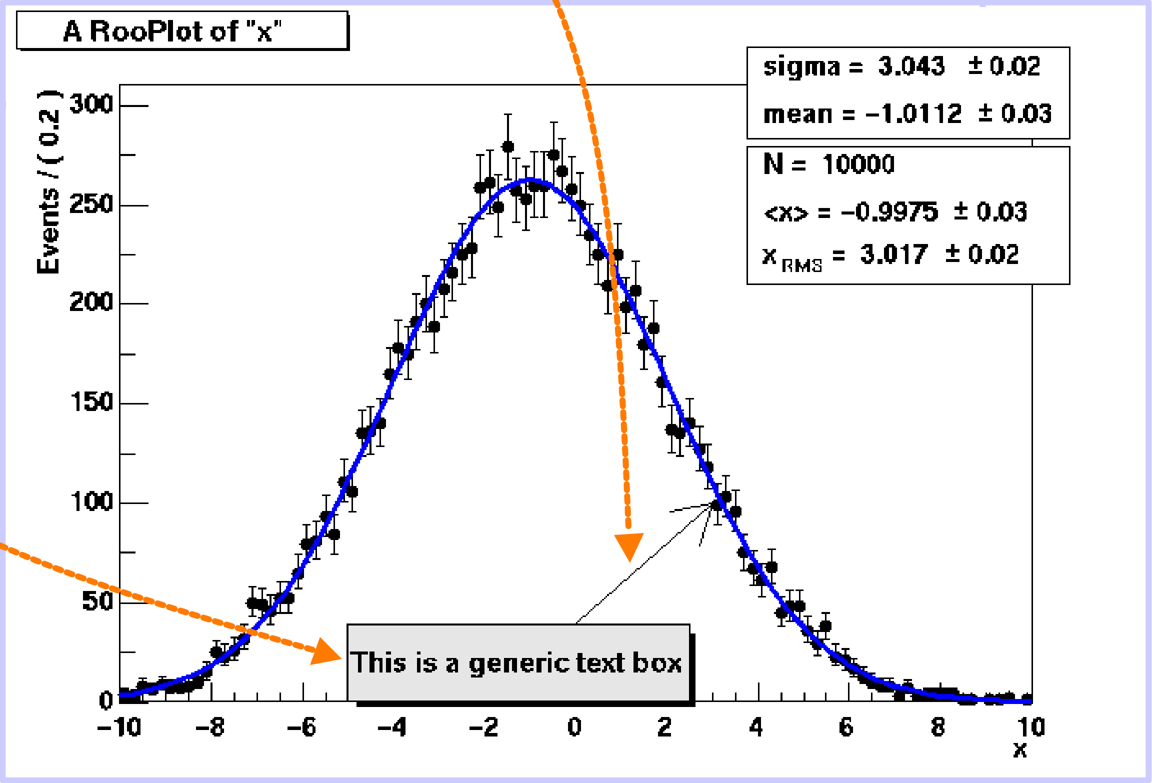
```
RooPlot* frame = x.frame() ;
data.plotOn(xframe) ;
pdf.plotOn(xframe) ;
pdf.paramOn(xframe,data) ;
data.statOn(xframe) ;
xframe->Draw() ;
```



Decoration

- Adding generic ROOT text boxes, arrows etc.

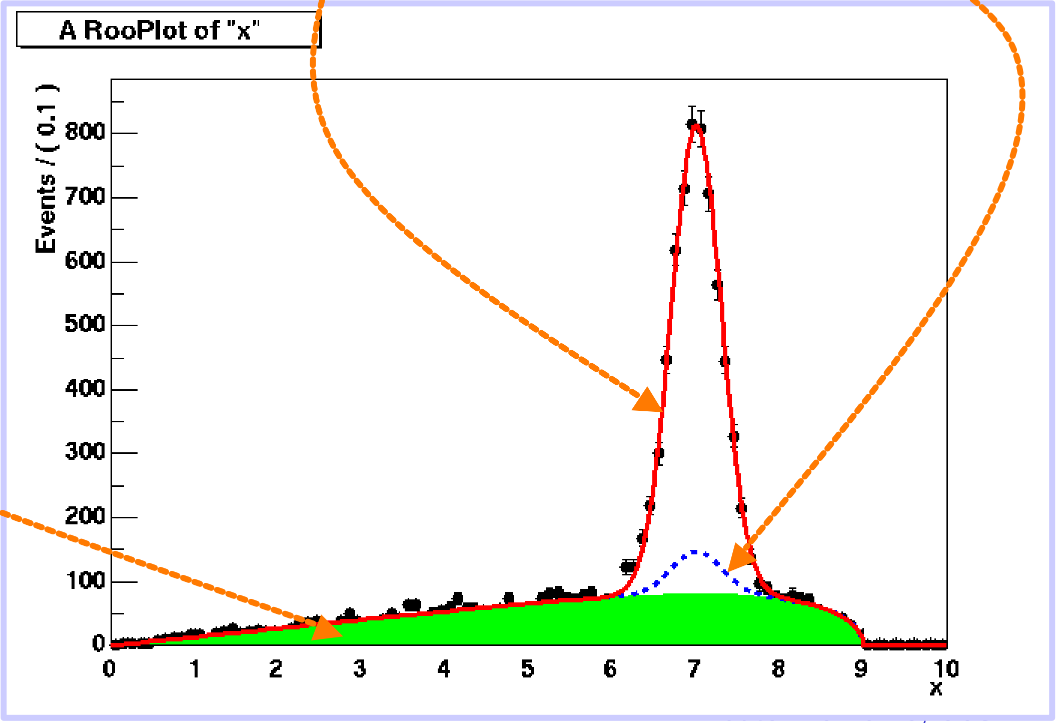
```
TPaveText* tbox = new TPaveText(0.3,0.1,0.6,0.2,"BRNDC");
tbox->AddText("This is a generic text box") ;
TArrow* arr = new TArrow(0,40,3,100) ;

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



Customization

- Changing colors and styles of histograms and curves

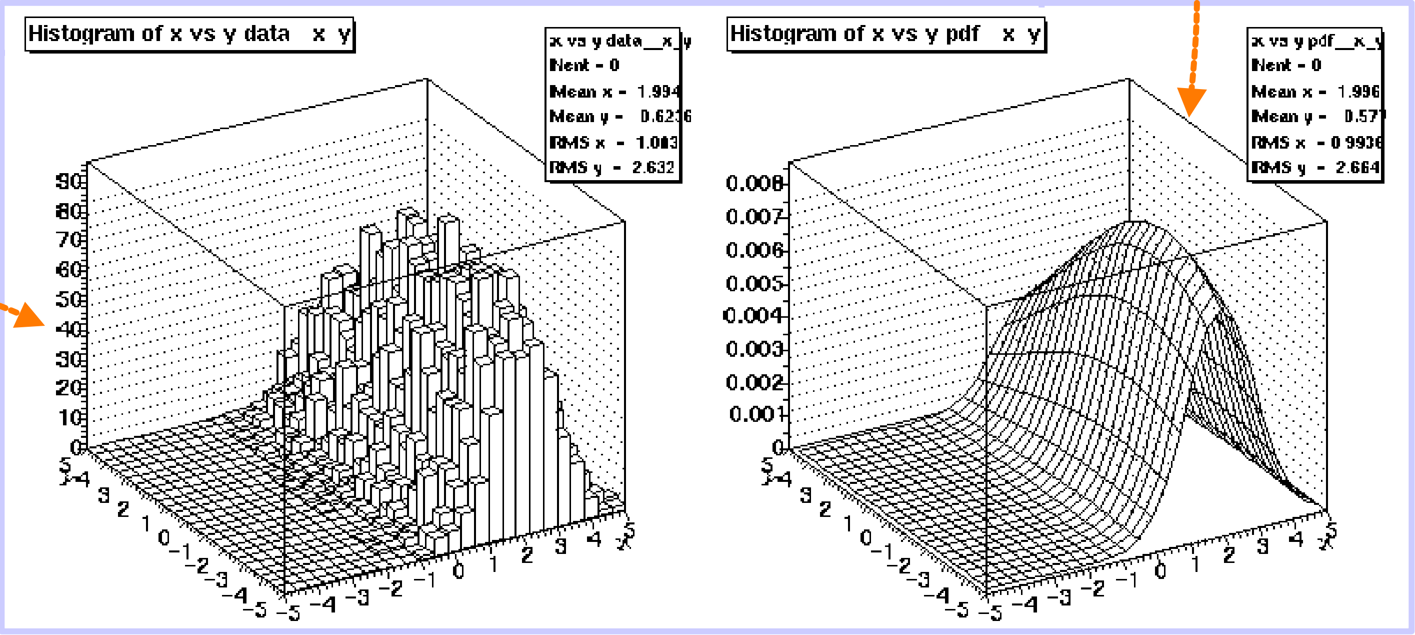
```
sum->plotOn(xframe,Components(RooArgSet(argus)),
                DrawOption("F“), FillColor(kGreen)) ;

sum->plotOn(xframe,Components(RooArgSet(argus,gauss2)),
                LineStyle(kDashed)) ;

sum->plotOn(xframe, LineColor(kRed) ;
```



Plotting in more than 2,3 dimensions

- No equivalent of RooPlot for >1 dimensions
  - Usually >1D plots are not overlaid anyway
  - Methods provided to produce 2/3D ROOT histograms from datasets and PDFs/functions

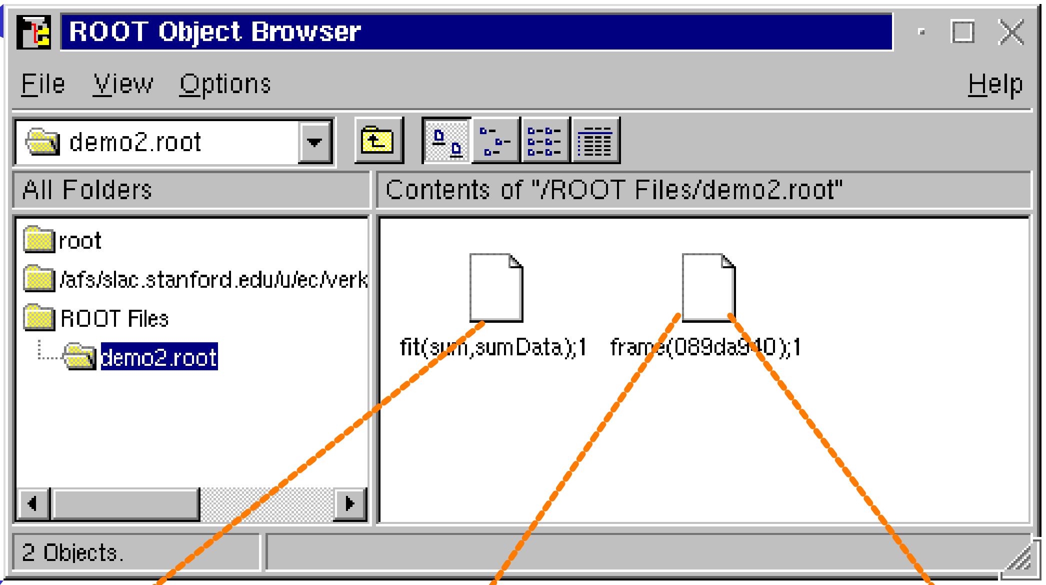
```
TH2* ph2 = x.createHistogram("x vs y pdf",y,0,0,0,bins) ;
prod.fillHistogram(ph2,RooArgList(x,y)) ;
ph2->Draw("SURF") ;

TH2* dh2 = x.createHistogram("x vs y data",y,0,0,0,bins) ;
data->fillHistogram(dh2,RooArgList(x,y)) ;
dh2->Draw("LEGO") ;
```



Persisting and reviving **RooPlots**

- Persisting

```
RooFitResult* r ;
RooPlot* xframe ;

Tfile f(“demo2.root”,
        ”RECREATE”) ;
r->Write() ;
xframe->Write() ;
f.Close() ;
```


ROOT Object Browser

File View Options Help

demo2.root

All Folders — Contents of "/ROOT Files/demo2.root"

root
/afs/slac.stanford.edu/u/ec/verk
ROOT Files
demo2.root

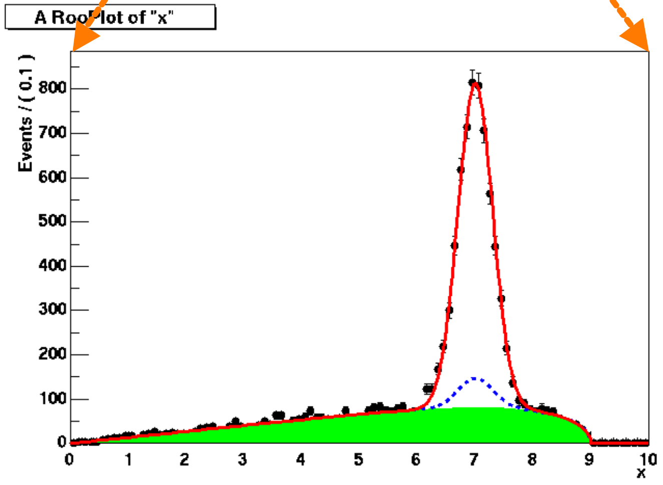
fit(sum,sumData);1
frame(089da940);1

2 Objects.

- Reviving

```
Tbrowswer tb ;

RooFitResult* r =
  f.Get(“fit(data,sum)”) ;
r->Print(“v”) ;
```

```
RooFitResult: min. NLL value: 1.6e+04, …

  Floating Parameter     FinalValue +/-  Error
  --------------------  --------------------------
              argpar     -4.6855e-01 +/-  7.11e-0
```



Wouter Verkerke, UCSB

Storing configuration data in ASCII files

- **RooArgLists** can be written to and read from ASCII file
  - Convenient to load initial values of fit parameters

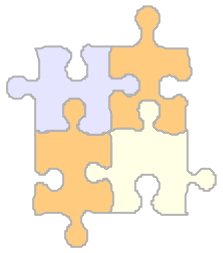
```
set.Print("v") ;
RooArgSet::parameters:
  1) RooRealVar::argpar : -0.468507 +/- 0.0711 (-0.0713, 0.0710) L(-2 - 0)
  2) RooRealVar::cutoff :  9.0000 C
  3) RooRealVar::g1frac :  0.30000 C
  4) RooRealVar::g2frac :  0.30652 +/- 0.00510 (-0.00509, 0.00511)
  5) RooRealVar::mean1  :  7.0022 +/- 0.00711 (-0.00712, 0.00710) L(0 - 10)
  6) RooRealVar::mean2  :  1.9971 +/- 0.00627 (-0.00628, 0.00626) L(0 - 10)
  7) RooRealVar::sigma  :  0.29803 +/- 0.00400 (-0.00396, 0.00403)
set.writeToFile("config.txt") ;
```

config.txt

```
argpar = -0.468507 +/- 0.0711 (-0.0713, 0.0710) L(-2 - 0)
cutoff =  9.0000  C
g1frac =  0.30000 C
g2frac =  0.30652 +/- 0.00510 (-0.00509, 0.00511)
mean1  =  7.0022  +/- 0.00711 (-0.00712, 0.00710) L(0 - 10)
mean2  =  1.9971  +/- 0.00627 (-0.00628, 0.00626) L(0 - 10)
sigma  =  0.29803 +/- 0.00400 (-0.00396, 0.00403)
```

```
set.readFromFile("config.txt") ;
```

Documentation



RooFit home page

Tutorial macros

Inline code documentation

How to get started / documentation

Starting point for all documentation is the RooFit homepage

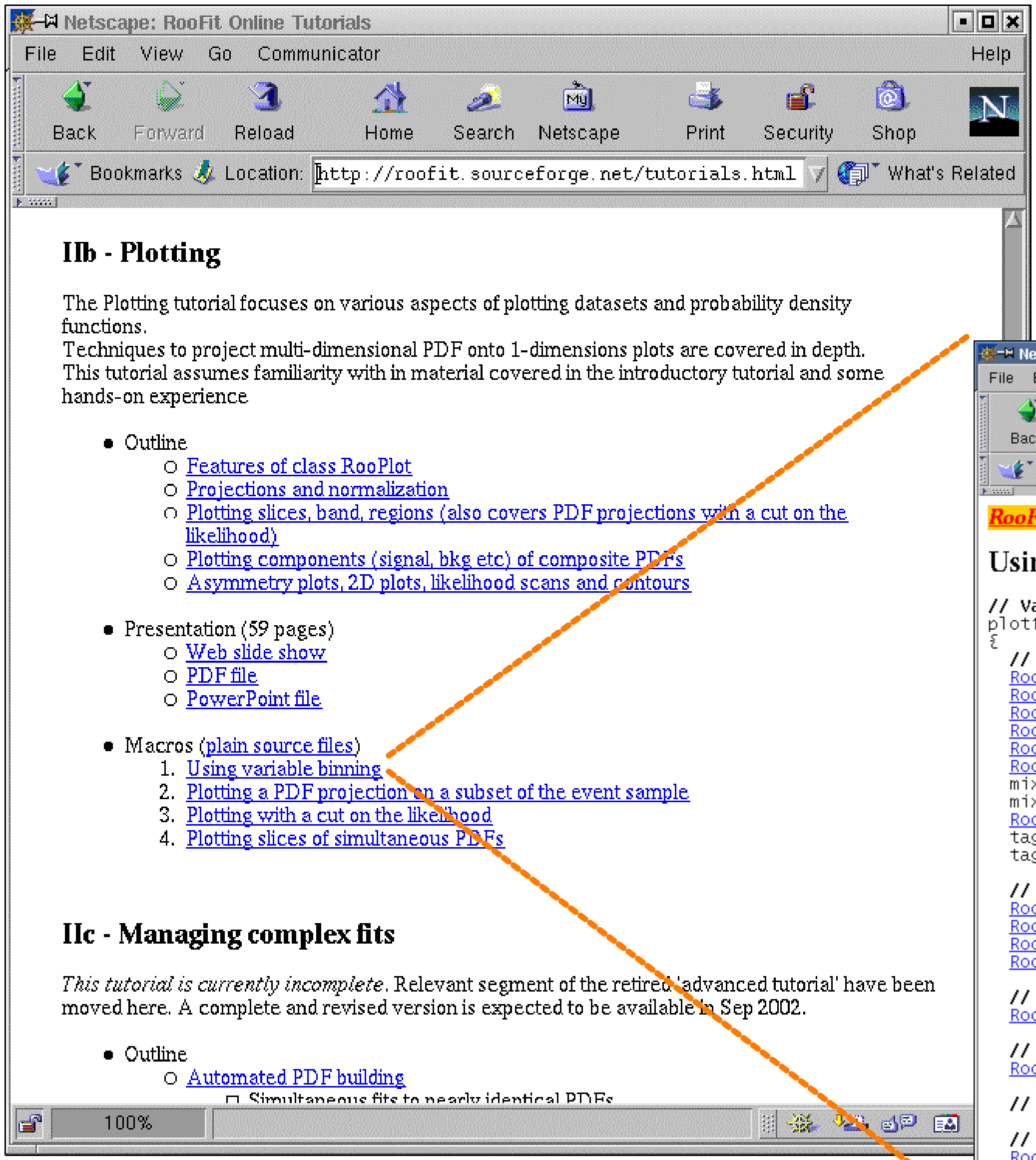
http://roofit.sourceforge.net

Online tutorials

demoXX.cc

Cover all topics of this presentation and more

macro references provided on slides


Netscape: RooFit Online Tutorials
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop
Bookmarks Location: http://roofit.sourceforge.net/tutorials.html
What's Related
100%

IIb - Plotting

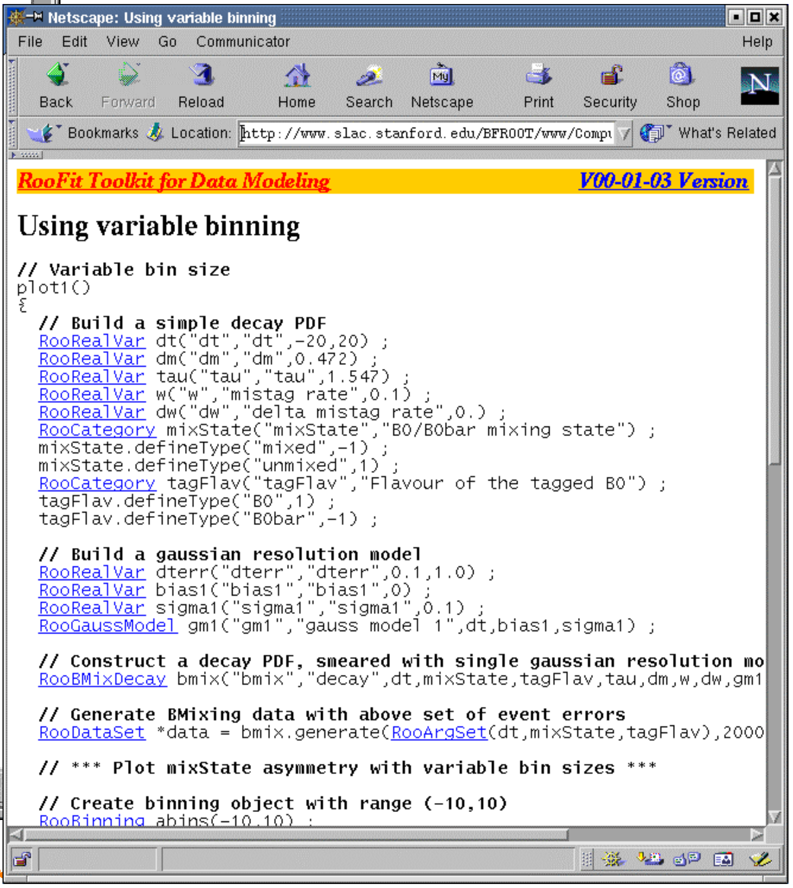
The Plotting tutorial focuses on various aspects of plotting datasets and probability density functions.
Techniques to project multi-dimensional PDF onto 1-dimensions plots are covered in depth.
This tutorial assumes familiarity with in material covered in the introductory tutorial and some hands-on experience

- Outline
  - Features of class RooPlot
  - Projections and normalization
  - Plotting slices, band, regions (also covers PDF projections with a cut on the likelihood)
  - Plotting components (signal, bkg etc) of composite PDFs
  - Asymmetry plots, 2D plots, likelihood scans and contours
- Presentation (59 pages)
  - Web slide show
  - PDF file
  - PowerPoint file
- Macros (plain source files)
  1. Using variable binning
  2. Plotting a PDF projection on a subset of the event sample
  3. Plotting with a cut on the likelihood
  4. Plotting slices of simultaneous PDFs

IIc - Managing complex fits

This tutorial is currently incomplete. Relevant segment of the retired 'advanced tutorial' have been moved here. A complete and revised version is expected to be available in Sep 2002.

- Outline
  - Automated PDF building
    - Simultaneous fits to nearly identical PDFs

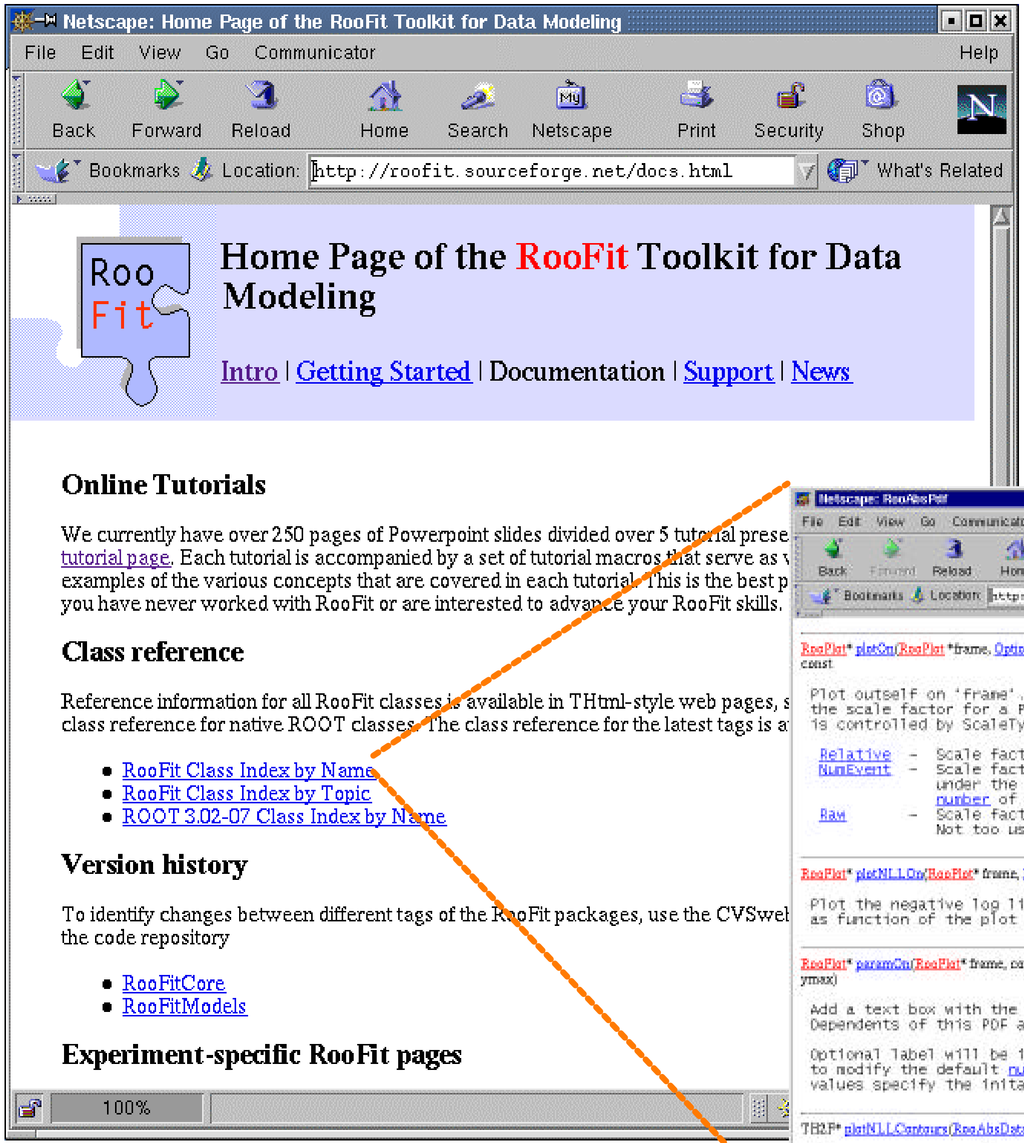

Netscape: Using variable binning
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop
Bookmarks Location: http://www.slac.stanford.edu/BFROOT/www/Compu
What's Related

RooFit Toolkit for Data Modeling — V00-01-03 Version

Using variable binning

```
// Variable bin size
plot1()
{
  // Build a simple decay PDF
  RooRealVar dt("dt","dt",-20,20) ;
  RooRealVar dm("dm","dm",0.472) ;
  RooRealVar tau("tau","tau",1.547) ;
  RooRealVar w("w","mistag rate",0.1) ;
  RooRealVar dw("dw","delta mistag rate",0.) ;
  RooCategory mixState("mixState","B0/B0bar mixing state") ;
  mixState.defineType("mixed",-1) ;
  mixState.defineType("unmixed",1) ;
  RooCategory tagFlav("tagFlav","Flavour of the tagged B0") ;
  tagFlav.defineType("B0",1) ;
  tagFlav.defineType("B0bar",-1) ;

  // Build a gaussian resolution model
  RooRealVar dterr("dterr","dterr",0.1,1.0) ;
  RooRealVar bias1("bias1","bias1",0) ;
  RooRealVar sigma1("sigma1","sigma1",0.1) ;
  RooGaussModel gm1("gm1","gauss model 1",dt,bias1,sigma1) ;

  // Construct a decay PDF, smeared with single gaussian resolution mo
  RooBMixDecay bmix("bmix","decay",dt,mixState,tagFlav,tau,dm,w,dw,gm1

  // Generate BMixing data with above set of event errors
  RooDataSet *data = bmix.generate(RooArgSet(dt,mixState,tagFlav),2000

  // *** Plot mixState asymmetry with variable bin sizes ***

  // Create binning object with range (-10,10)
  RooBinning abins(-10,10) ;
```

HTML class documentation


Netscape: Home Page of the RooFit Toolkit for Data Modeling
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Shop
Bookmarks Location: http://roofit.sourceforge.net/docs.html What's Related
Roo Fit
100%

Home Page of the RooFit Toolkit for Data Modeling

Intro | Getting Started | Documentation | Support | News

Online Tutorials

We currently have over 250 pages of Powerpoint slides divided over 5 tutorial prese
tutorial page. Each tutorial is accompanied by a set of tutorial macros that serve as v
examples of the various concepts that are covered in each tutorial. This is the best p
you have never worked with RooFit or are interested to advance your RooFit skills.

Class reference

Reference information for all RooFit classes is available in THtml-style web pages, s
class reference for native ROOT classes. The class reference for the latest tags is a

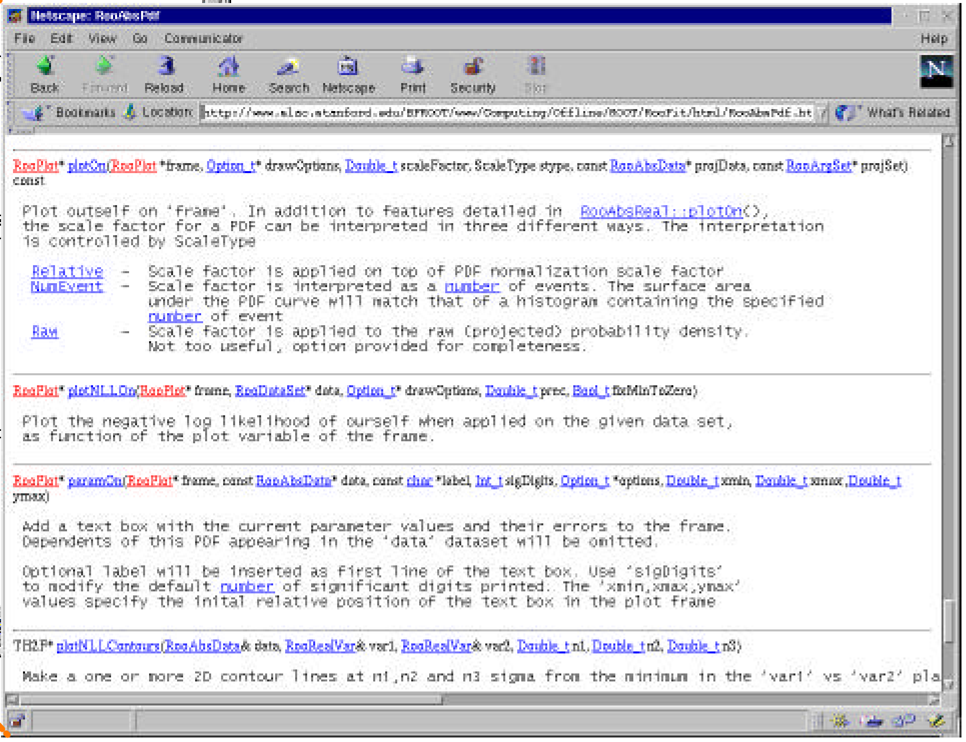
- RooFit Class Index by Name
- RooFit Class Index by Topic
- ROOT 3.02-07 Class Index by Name

Version history

To identify changes between different tags of the RooFit packages, use the CVSwel
the code repository

- RooFitCore
- RooFitModels

Experiment-specific RooFit pages


Netscape: RooAbsPdf
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security Stop
Bookmarks Location: http://www.slac.stanford.edu/BFROOT/www/Computing/Offline/ROOT/RooFit/html/RooAbsPdf.ht What's Related

RooPlot* plotOn(RooPlot *frame, Option_t* drawOptions, Double_t scaleFactor, ScaleType stype, const RooAbsData* projData, const RooArgSet* projSet) const

```
Plot outself on 'frame'. In addition to features detailed in  RooAbsReal::plotOn(),
the scale factor for a PDF can be interpreted in three different ways. The interpretation
is controlled by ScaleType

 Relative   -   Scale factor is applied on top of PDF normalization scale factor
 NumEvent   -   Scale factor is interpreted as a number of events. The surface area
                under the PDF curve will match that of a histogram containing the specified
                number of event
 Raw        -   Scale factor is applied to the raw (projected) probability density.
                Not too useful, option provided for completeness.
```

RooPlot* plotNLLOn(RooPlot* frame, RooDataSet* data, Option_t* drawOptions, Double_t prec, Bool_t fixMinToZero)

```
Plot the negative log likelihood of ourself when applied on the given data set,
as function of the plot variable of the frame.
```

RooPlot* paramOn(RooPlot* frame, const RooAbsData* data, const char *label, Int_t sigDigits, Option_t *options, Double_t xmin, Double_t xmax ,Double_t ymax)

```
Add a text box with the current parameter values and their errors to the frame.
Dependents of this PDF appearing in the 'data' dataset will be omitted.

Optional label will be inserted as first line of the text box. Use 'sigDigits'
to modify the default number of significant digits printed. The 'xmin,xmax,ymax'
values specify the inital relative position of the text box in the plot frame
```

TH2F* plotNLLContours(RooAbsData& data, RooRealVar& var1, RooRealVar& var2, Double_t n1, Double_t n2, Double_t n3)

```
Make a one or more 2D contour lines at n1,n2 and n3 sigma from the minimum in the 'var1' vs 'var2' pla
```

Same format as ROOT native class docs

Prebuilt version on web, easy to build your own for very latest version