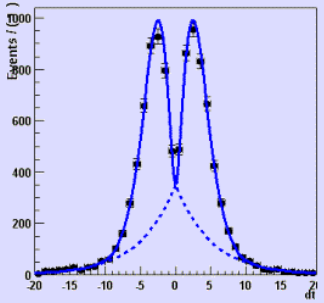
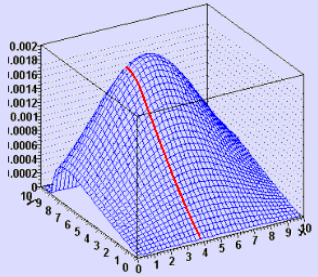
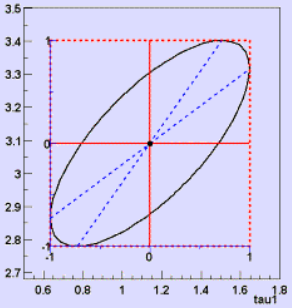






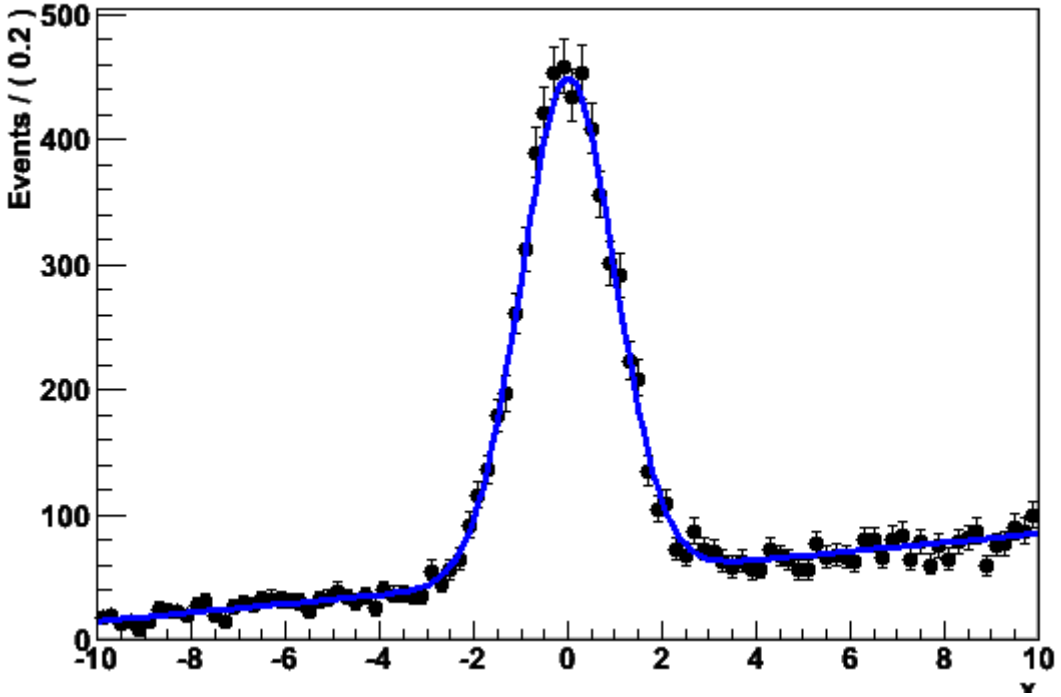
RooFit

A tool kit for data modeling in ROOT

Wouter Verkerke (NIKHEF)
David Kirkby (UC Irvine)

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

The situation at BaBar six years ago…

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

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

- 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

A recent example

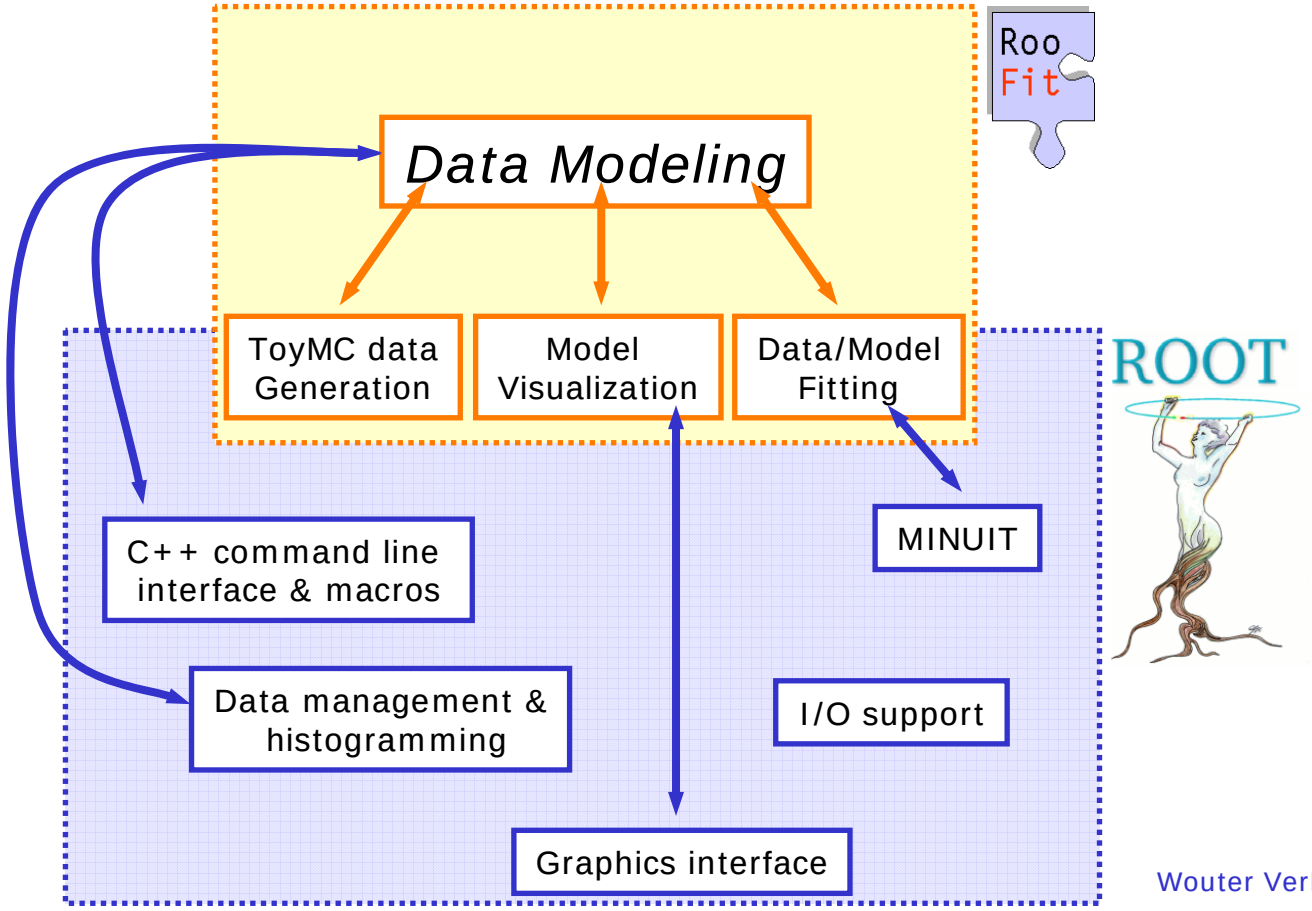
- Initial approach BaBar: write it from scratch (in FORTRAN!)
  - Does its designated job quite well, but took a long time to develop
  - Possible because sin(2β) effort supported by O(50) people.
  - Optimization of ML calculations hand-coded → error prone and not easy to maintain
  - Difficult to transfer knowledge/code from one analysis to another.

- A better solution: A modeling language in C++ that integrates seamlessly into ROOT
  - Recycle code *and* knowledge

- Development of RooFit package
  - Started 5 years ago.
  - Very successful → virtually everybody in BaBar uses it → now in standard ROOT distribution

What is RooFit

- A data modeling language to facilitate medium complex to very complex fits
  - Addition to ROOT – (Almost) no overlap with existing ROOT functionality



Data modeling – OO representation

- TF1s single line ASCII math expression quickly becomes limiting factor when writing on-trivial functions
  - Idea: represent *each* math symbols with C++ object

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

  - Result: 1 line of code per symbol in a function (the C++ constructor) rather than 1 line of code per function

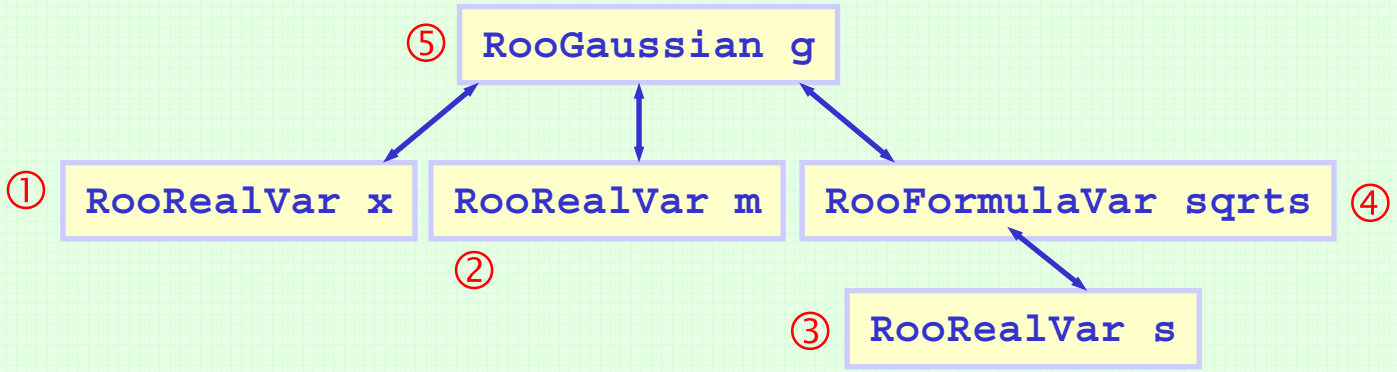
Data modeling – Constructing composite objects

- Straightforward correlation between mathematical representation of formula and RooFit code

Math

$$gauss(x, m, \sqrt{s})$$

RooFit diagram

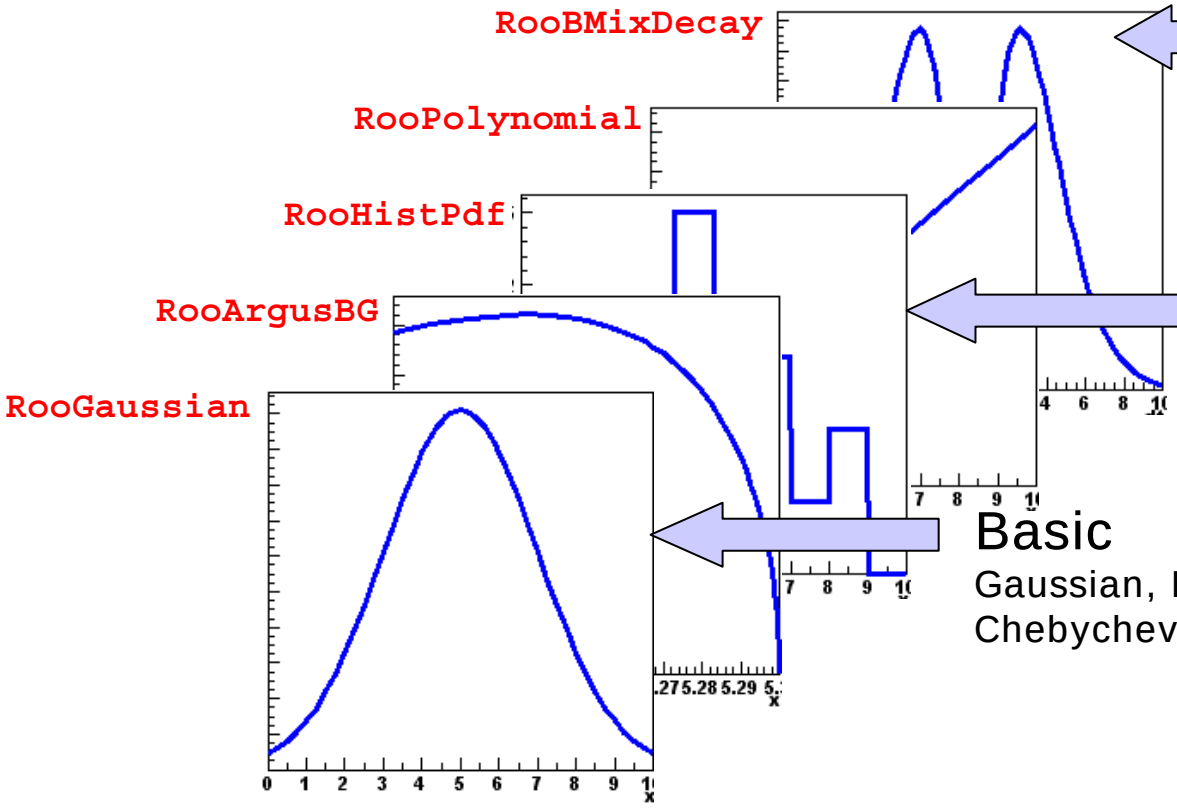


RooFit code

```
① RooRealVar x("x","x",-10,10) ;
② RooRealVar m("m","mean",0) ;
③ RooRealVar s("s","sigma",2,0,10) ;
④ RooFormulaVar sqrts("sqrts","sqrt(s)",s) ;
⑤ RooGaussian g("g","gauss",x,m,sqrts) ;
```

Model building – (Re)using standard components

- RooFit provides a collection of compiled standard PDF classes



Physics inspired
ARGUS,Crystal Ball,
Breit-Wigner, Voigtian,
B/D-Decay,….

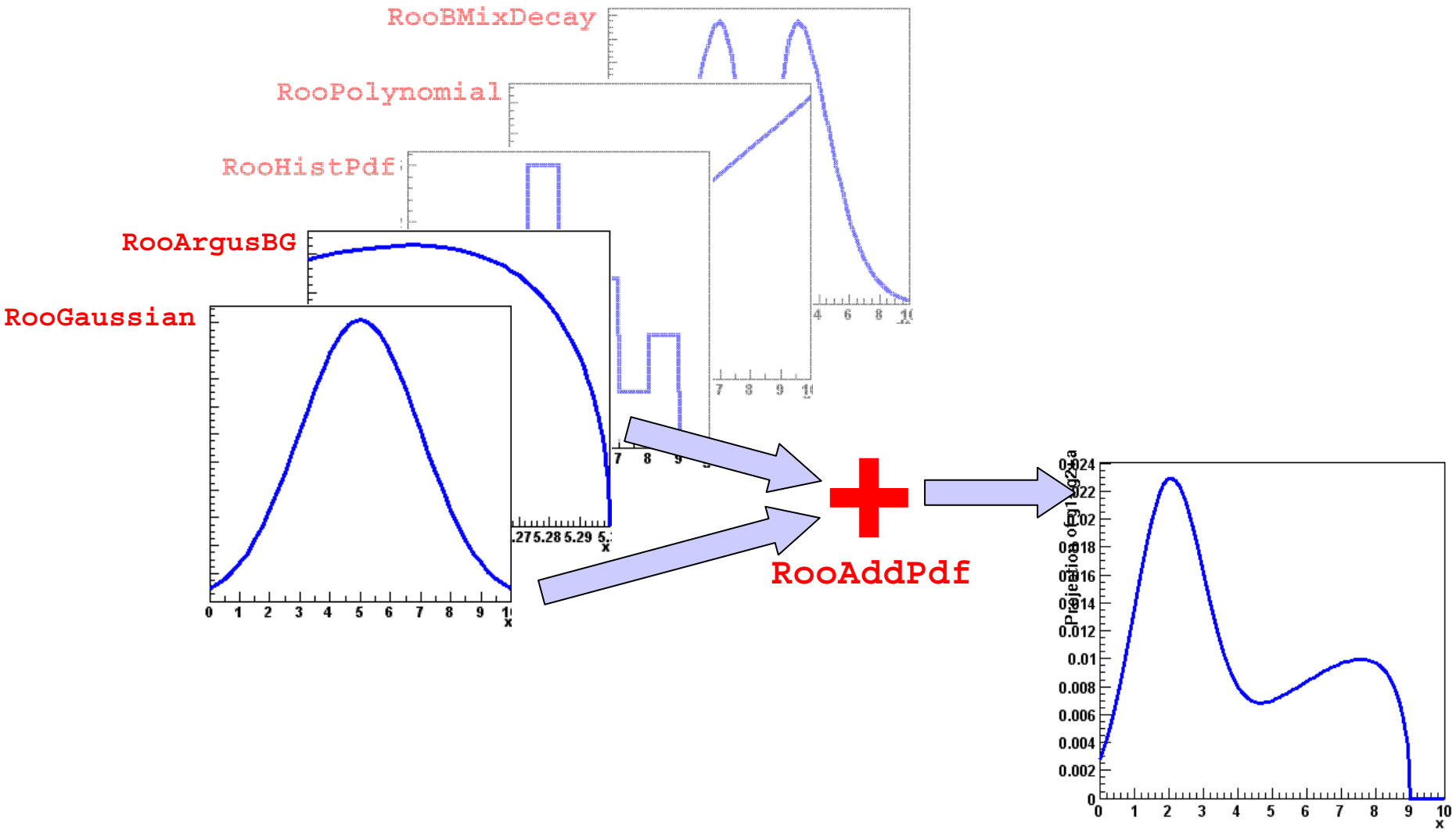
Non-parametric
Histogram, **KEYS**

Basic
Gaussian, Exponential, Polynomial,…
Chebychev polynomial

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

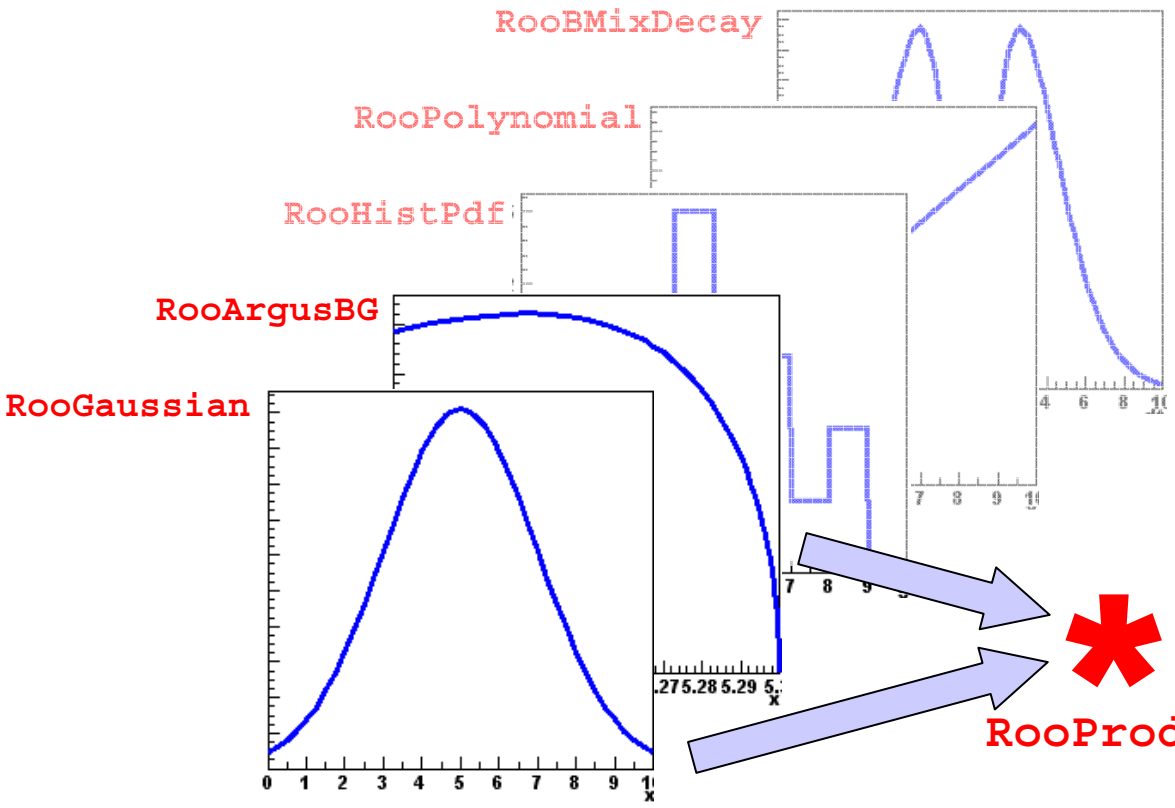
Model building – (Re)using standard components

- Most physics models can be composed from ‘basic’ shapes



Model building – (Re)using standard components

- Most physics models can be composed from ‘basic’ shapes

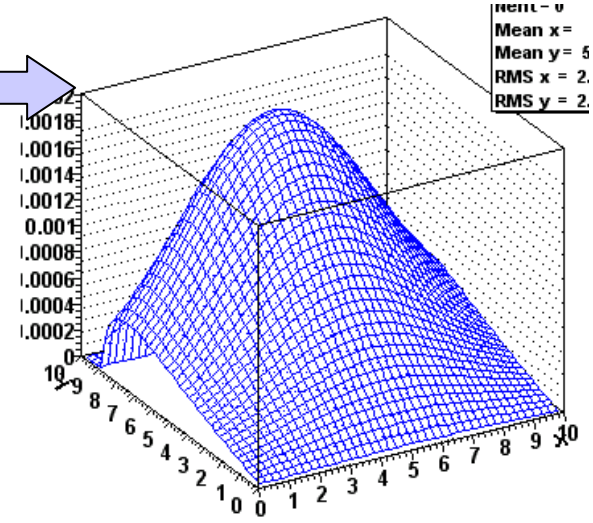



```
RooProdPdf h("h","h",
             RooArgSet(f,g))
```

$$h(x,y) = f(x) \cdot g(y)$$

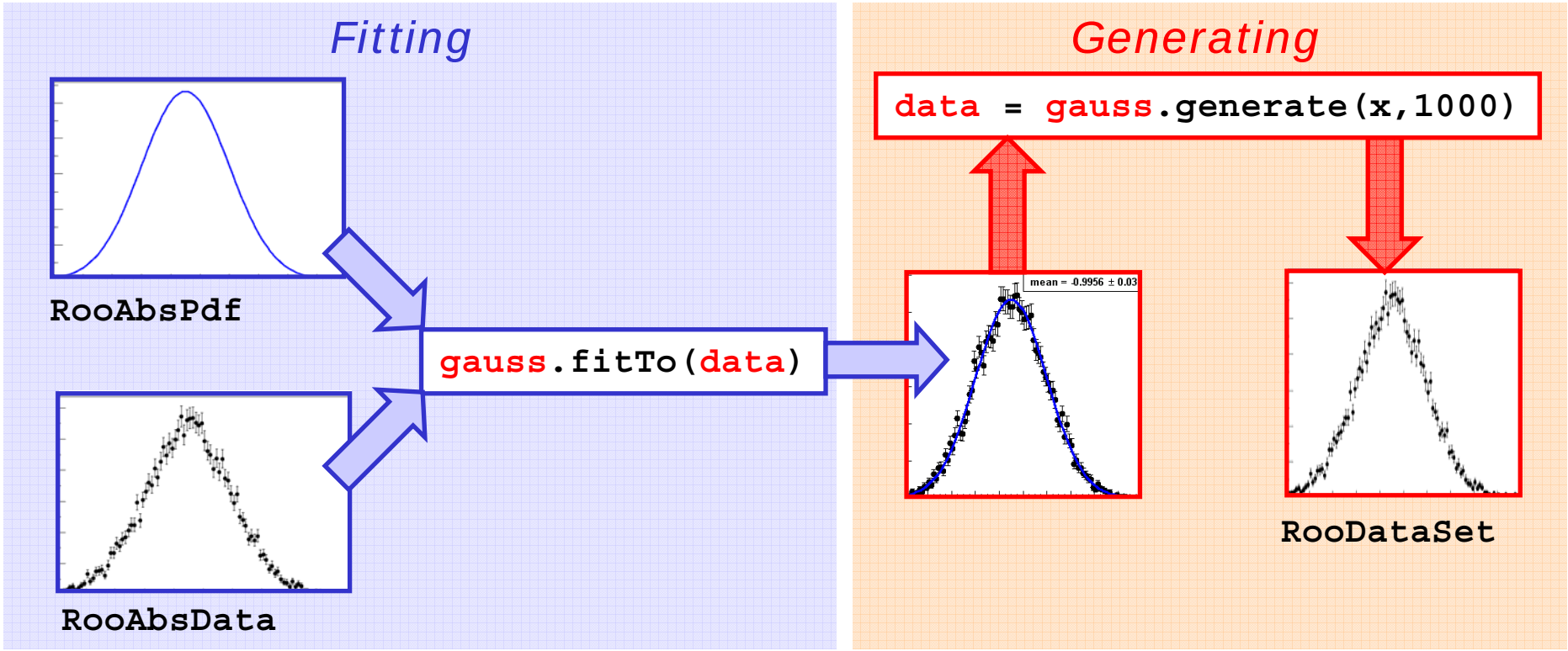
```
RooProdPdf k("k","k",g,
             Conditional(f,x))
```

$$k(x,y) = f(x \mid y) \cdot g(y)$$



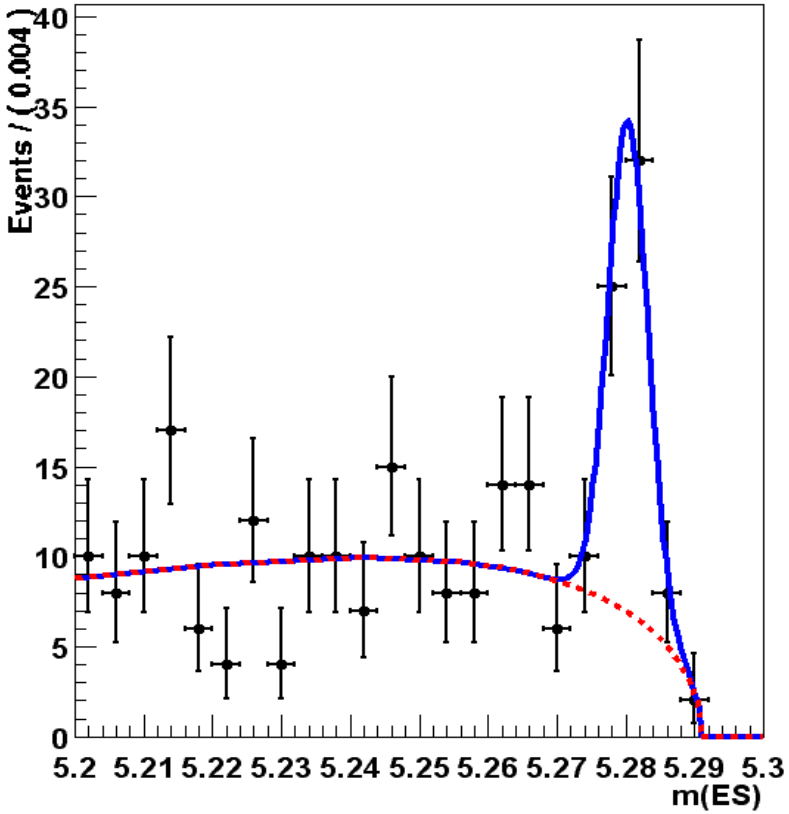
Using models - Overview

- *All* RooFit models provide *universal and complete fitting* and Toy Monte Carlo *generating* functionality
  - Model complexity only limited by available memory and CPU power
  - Fitting/plotting a 5-D model as easy as using a 1-D model
  - Most operations are one-liners



Using models – Plotting

- Model visualization geared towards 'publication plots' not interactive browsing → emphasis on 1-dimensional plots
- Simplest case: plotting a 1-D model over data
  - *Modular structure of composite p.d.f.s allows easy access to components for plotting*
  - *Can show Poisson confidence intervals instead of sqrt(N) errors*




```
RooPlot* frame = mes.frame() ;
data->plotOn(frame) ;
pdf->plotOn(frame) ;

pdf->plotOn(frame,Components("bkg"))

frame->Draw() ;
```

Can store plot with data and all curves as single object

RooFit design philosophy

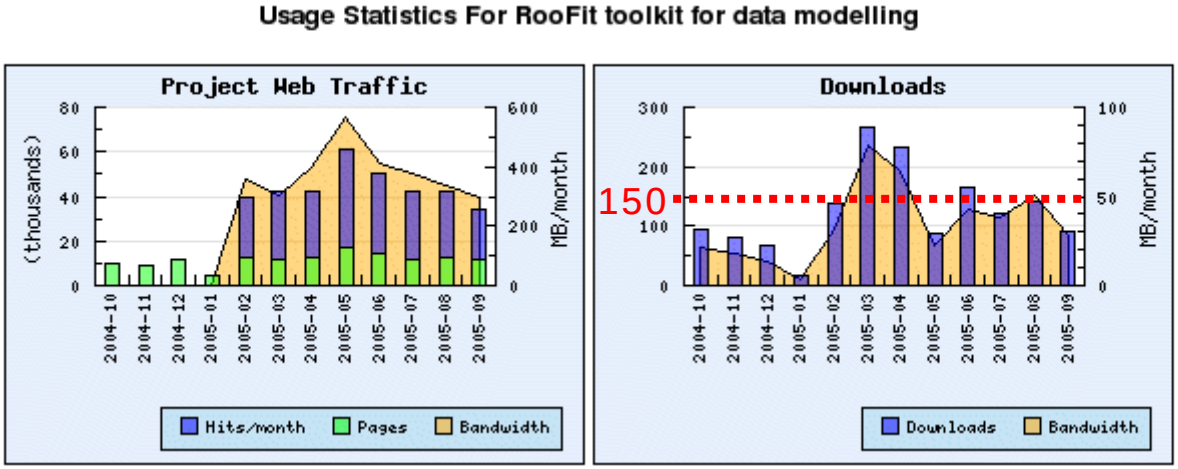
- No ‘arbitrary’ implementation restricted limitations
  - A `RooProdPdf` can multiply any number of PDFs of *any* type
  - A `RooNumConvPdf` can convolve *any two* PDFs
  - `pdf.fitTo()` is fully functional on *any* PDFs
  - `pdf.generate()` can generate *any* number of observables from *any* PDF
  - `pdf.plotOn()` works for *any* PDF

- Achieve complexity through composition
  - Try to find the minimum number of building blocks and operators that allow to do everything you want
  - Example: decay ⊗ (gauss1 + gauss2)
    - No need for DoubleGauss resolution model as operator class RooAddModel solves this job (and many other ones)

- Exact optimizations for speed are a computing problem, not a physics problem
  - An exact optimization is really an algorithm. RooFit can applies these for you consistently and effortlessly for you in the best possible way

Development history and use of RooFit

- RooFit started as RooFitTools (presented at ROOT2001) in late 1999 for the BaBar Collaboration
  - Original design was rapidly stretched to its limits
- Started comprehensive redesign early 2001
  - New design was released to BaBar users in Oct 2001 as RooFit
- RooFit released on SourceForge in Sep 2002
  - **http://roofit.sourceforge.net**
- Vibrant user community:
  - **Averaging 150 downloads per month (in last 12 months), 40K web hits per month!**
  - Additional downloads via CVS not measured.
  - BaBar use not included in above as they have a copy in their own CVS/release structure



Scientific output using RooFit

- Selection of BaBar publications in 2004 using RooFit
  - Improved Measurement of the CKM angle alpha using $B^0 \to \rho^+\rho^-$
  - Measurement of branching fractions and charge asymmetries in B^+ decays to $\eta\pi^+$, ηK^+, $\eta\rho^+$ and $\eta'\pi^+$, and search for B^0 decays to ηK^0 and $\eta\omega$
  - Branching Fraction and CP Asymmetries in $B^0 \to K_S K_S K_S$
  - Measurement of CP Asymmetries in $B^0 \to \phi K^0$ and $B^0 \to K^+K^-K^0_S$
  - Measurement of Branching Fractions and Time-Dependent CP-Violating Asymmetries in $B \to \eta'$ K Decays
  - Improved Measurement of Time-Dependent CP Violation in B0 to (ccbar)K^0 Decays ('sin2β')
  - Measurements of the Branching Fraction and CP-Violating Asymmetries in $B0 \to f^0(980)K_S$ Decays
  - Measurement of Time-dependent CP-Violating Asymmetries in $B^0 \to K^*\gamma$, $K^* \to K_S\pi^0$ Decays
  - Study of the decay $B^0 \to \rho+\rho-$ and constraints on the CKM angle alpha.
  - Measurement of CP-violating Asymmetries in $B^0 \to K^0_S\pi^0$ Decays
  - Measurement of Time-Dependent CP Asymmetries in $B^0 \to \phi K^0$
  - Search for $B^{+/-} \to [K^{-/+}\ \pi^{+/-}]_D\ K^{+/-}$ and upper limit on the b→u amplitude in $B^{+/-} \to DK^{+/-}$
  - Limits on the Decay-Rate Difference of Neutral B Mesons and on CP, T, and CPT Violation in B^0B^0bar Oscillations

Development history and use of RooFit

- RooFit v2 in June 2005
  - Evolution of design, no major changes
  - Make it easier to also do ‘simple’ modeling problems (less focus on B physics)

- RooFit v2.05 bundled with ROOT v5 distribution
  - Code continues to be developed on SourceForge
  - Each ROOT5 release contains a zipped tar file with a RooFit release and includes the necessary make files to build it as part of the ROOT system
  - Simplifies access for new & old users: libraries are readily available and compile on all ROOT supported platforms (including MacOS, native Windows)

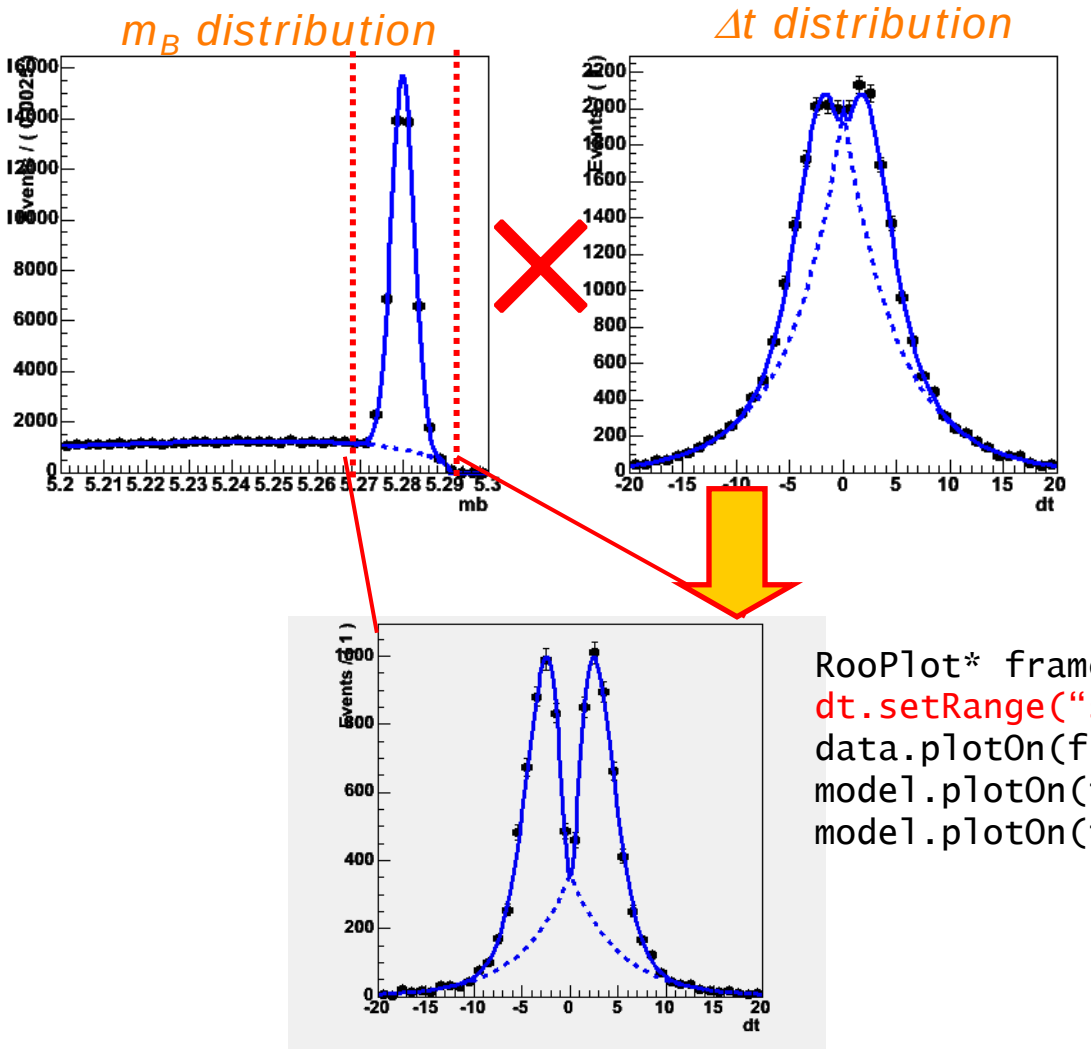
- Big Project (~10% of ROOT5)
  - 56K lines of C++ source code
  - 177 C++ classes

Summary of developments for v2.05

- General code maintenance
  - Extensive cleaning of code: Code now compiles cleanly on all ROOT supported platform
  - Packaging as ROOT module (Module.mk file etc)
- Design of interface evolving gradually
  - Basic design concept provide solid foundation
  - Most new features make RooFit easier to use: implementation usually achieved by removal of limitation in existing interface rather than adding a new interface
- New/Enhanced features
  - New numeric convolution operator class, new numeric integration methods,
    - improved interface to control numeric integration methods and parameters
  - Concept of named ranges associated with variables to support complex views, projections and integral ratios in a natural way
  - Code factory that simplifies use writing compiled classes on the fly using ROOT ACLiC
  - Latex output for RooFit tables and lists
  - Improved manipulation of RooPlot contents
  - Roll-out of 'named argument' interface for most major functions

Selection of recent improvements – named ranges

- Easy to project slices of both data *and functions*
  - *Slices of function not generally easy to calculate, but RooFit will handle any p.d.f*

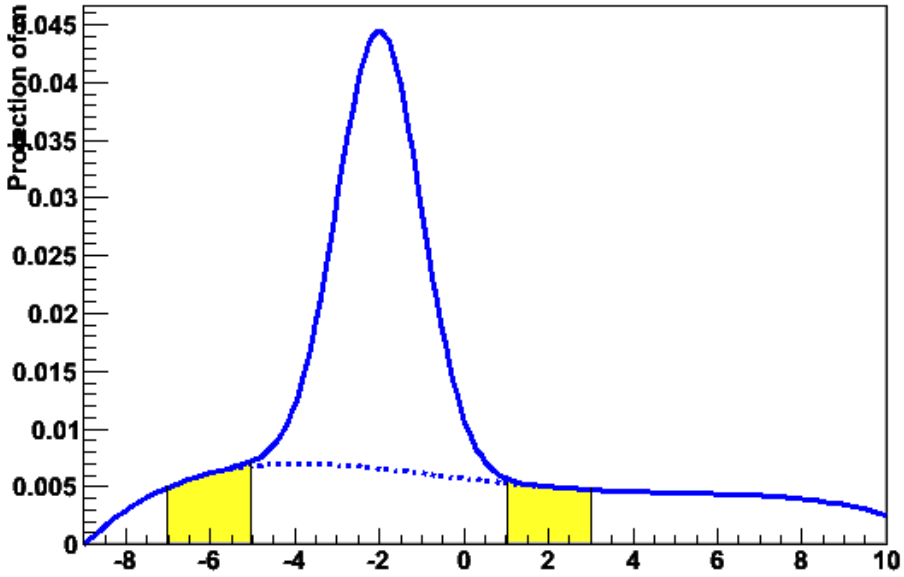



```
RooPlot* frame = dt.frame() ;
data.plotOn(frame) ;
model.plotOn(frame) ;
model.plotOn(frame,Components("bkg")) ;
```

```
RooPlot* frame = dt.frame() ;
dt.setRange("sel",5.27,5.30) ;
data.plotOn(frame,CutRange("sel")) ;
model.plotOn(frame,ProjectionRange("sel"));
model.plotOn(frame,ProjectionRange("sel"),
                   Components("bkg")) ;
```

Selection of recent improvements – named ranges

- Another example using named ranges:
  - Calculate ratio of bkg in dual sideband over bkg in signal region




```
x.setRange("sblo",-7,-5) ;
x.setRange("sig",-5,1) ;
x.setRange("sbhi",1,3) ;
RooAbsReal* fracSB = bkg.createIntegral(x,Range("sblo,sbhi")) ;
RooAbsReal* fracSig = bkg.createIntegral(x,Range("sig")) ;
cout << "sb/sig ratio = " << fracSB->getVal()/fracSig->getVal() ;
```

Documentation effort

- Major effort now ongoing in documentation
  - Current documentation set of PPT presentations for BaBar collaboration.
  - Covers most features but somewhat specific to B-physics and presentation style does not allow for in-depth coverage of important details

- New two-prong approach:
  - ROOT-style printed *Users Manual*, a pedagogical document with a reference section (~100 pages total due by Dec 2005)
  - Online WIKI documentation for practical solution, examples ranging from simple to complex (under development)

Documentation – Shapshot of 'Users Guide'

Convolving a p.d.f. or function with another p.d.f.

Introduction

If you are modeling distribution of an experimental observable you are sometimes faced with a situation where you should explicitly take into account the deformation of the expected signal distributed due to the finite detector resolution. This issue becomes particularly important when the detector resolution is comparable to the structure (width) of your expected signal.

In general, the observed distribution is described by the convolution of your physics model $T(x,a)$ and your detector response function $R(x,b)$

$$M(x,a,b) = T(x,a) \otimes R(x,b) = \int_{-\infty}^{+\infty} T(x,a) R(x-x',b) dx'$$

In practice the detector response function R is often a Gaussian, or a superposition of Gaussians. Figure 12 illustrates the effect of a Gaussian resolution model R with three different widths on a Breit-Wigner function.

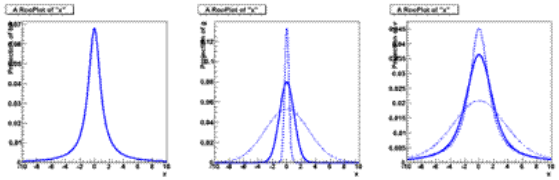


Figure 12 – left: Breit-Wigner, middle Gaussian (σ=0.3, 1, 3) right: Breit-Wigner convolved with Gaussian

You can see from Figure 12 that if R is narrow with respect to T (dotted line), the convolution T⊗R is well approximated by T. If R is wide with respect to T (dashed line), the convolution T⊗R is well approximated by R, therefore modeling your signal p.d.f. explicitly as T⊗R is usually only important if both are comparable width. This is a good thing, since calculation of integral that represents T⊗R is generically quite difficult. The normalization condition for p.d.f.s. adds one further difficulty as the final quantity acquires a double integral in the denominator.

$$M(x,a,b) = \frac{\int_{-\infty}^{\infty} T(x,a) R(x-x',b) dx'}{\int_{x_{min}}^{x_{max}} \int_{-\infty}^{\infty} T(x,a) R(x-x',b) dx' dx}$$

You are best off if you don't need to perform this calculation, but sometimes you just have to. In the remainder of this section we'll explain how you can deal with convolved p.d.f.s in RooFit.

Analytical versus numeric convolution

A precise and fast calculation of the convolution integral is essential as p.d.f.s are evaluated a large number of times in the course of a fit. Because of that an analytical expression for the convolution

integral is therefore strongly preferred. Unfortunately this is not always possible, and a numeric calculation of the integral must sometimes be used as fallback solution.

Analytical convolution

RooFit does not find analytical expressions for convolution integrals for you, but helps you to implement them in a generic and reusable way. It does this by defining two specialize sub-classes of p.d.f.s.: convolvable p.d.f.s, which implement $T(x,a)$ and resolution models, which implement $R(x,b)$. You can combine any R and T at *runtime* into a $M(x,a,b)=T(x,a) \otimes R(x,b)$ so you are quite flexible in your choice of convolutions when you build your model. RooFit provides out-of-the-box the following convolvable p.d.f.s

Class Name	Description
RooDecay	Decay function: exp(-|t|/τ)
RooBMixDecay	B decay with mixing
RooBCPEffDecay	B decay with CP violation parameterized as sin(2b) and |l|
RooBCPGenDecay	B decay with CP violation parameterized S and C
RooNonCPEigenDecay	B decay to non-CP eigenstates with CP violation
RooBDecay	Generic B decay with mixing, CP violation, CPT violation

And it provides the following resolution models.

Name	Functional form	Class name
Gauss	$\exp\left(-0.5\left(\frac{x-m}{s}\right)^2\right)$	`RooGaussModel(name,title,x,m,s)`
Gauss⊗Exp	$\exp\left(-0.5\left(\frac{x-m}{s}\right)^2\right) \otimes \exp(-x/\tau)$	`RooGExpModel(name,title,x,m,s,tau)`
Truth	$\delta(x)$	`RooTruthModel(name,title,x)`
Composite	$\sum_{i=1,n-1} f_i R_i(x,\alpha) + \left(1 - \sum_{i=1,n-1} f_i\right) R_n(x,\alpha)$	`RooAddModel(name,title,Rlist,flist)`

To construct an analytically convolved p.d.f. pass one of the `RooResolutionModel` implementations to the construct of a convolvable p.d.f. In the example below we construct a decay function convolved with a Gaussian resolution model:

```
RooRealVar x("x","x",-10,10) ;

RooRealVar mean("mean","mean",0) ;
RooRealVar sigma("sigma","sigma",1) ;
RooGaussModel gaussm("gaussm",x,mean,sigma) ;

RooRealVar tau("tau","lifetime",1.54) ;
RooDecay model("model","decay (x) gauss",x,tau,gaussm) ;

// --- Plot decay (x) gauss ---
RooPlot* frame = x.frame() ;
model.plotOn(frame) ;
```

Documentation – Reference section of guide

RooAbsCollection& other)	objects with the same name in the same order. If this is not the case warnings will be printed. If a single sibling list is specified, 3 columns will be output: the (common) name, the value of this list and the value in the sibling list. Multiple sibling lists can be specified by repeating the Sibling() command.
Format(const char* str)	Classic format string, provided for backward compatibility
Format(...)	Formatting arguments, details are given below
OutputFile(const char* fname)	Send output to file with given name rather than standard output

The Format(const char* what,...)has the following structure

const char* what	Controls what is shown. "N" adds name, "E" adds error, "A" shows asymmetric error, "U" shows unit, "H" hides the value
FixedPrecision(int n)	Controls precision, set fixed number of digits
AutoPrecision(int n)	Controls precision. Number of shown digits is calculated from error + n specified additional digits (1 is sensible default)
VerbatimName(Bool_t flag)	Put variable name in a \verb+ +clause.

Example use:

```
list.printLatex(Columns(2), Format("NEU",AutoPrecision(1),VerbatimName()) ) ;
```

Generating toy Monte Carlo datasets – RooAbsPdf::generate()

Usage example: `RooDataSet* data = pdf.generate(x,...) ;`

Generate a new dataset containing the specified variables with events sampled from our distribution. Generate the specified number of events or expectedEvents() if not specified.

Any variables of this PDF that are not in whatVars will use their current values and be treated as fixed parameters. Returns zero in case of an error. The caller takes ownership of the returned dataset.

The following named arguments are supported

Verbose(Bool_t flag)	Print informational messages during event generation
NumEvent(int nevt)	Generate specified number of events
Extended()	The actual number of events generated will be sampled from a Poisson distribution with mu=nevt. For use with extended maximum likelihood fits
ProtoData(const RooDataSet& data, Bool_t randOrder)	Use specified dataset as prototype dataset. If randOrder is set to true the order of the events in the dataset will be read in a random order the order of the events in the dataset will be read in a random order number of events in the prototype dataset

If ProtoData() is used, the specified existing dataset as a prototype: the new dataset will contain the same number of events as the prototype (unless otherwise specified), and any prototype variables not in whatVars will be copied into the new dataset for each generated event and also used to set our PDF parameters.

The user can specify a number of events to generate that will override the default. The result is a copy of the prototype dataset with only variables in whatVars randomized. Variables in whatVars that are not in the prototype will be added as new columns to the generated dataset.

Creating integrals of functions– RooAbsReal::createIntegral()

Usage example: `RooAbsReal* intOfFunc = func.createIntegral(x,...) ;`

Create an object that represents the integral of the function over one or more observables listed in iset

The actual integration calculation is only performed when the return object is evaluated. The name of the integral object is automatically constructed from the name of the input function, the variables it integrates and the range integrates over

The following named arguments are accepted

NormSet(const RooArgSet&)	Specify normalization set, mostly useful when working with PDFS
NumIntConfig(const RooNumIntConfig&)	Use given configuration for any numeric integration, if necessary
Range(const char* name)	Integrate only over given range. Multiple ranges may be specified by passing multiple Range() arguments

Automated fit studies – RooMCStudy

Usage example: `RooMCStudy mgr(model,observables,...) ;`

Construct Monte Carlo Study Manager. This class automates generating data from a given PDF, fitting the PDF to that data and accumulating the fit statistics.

The constructor accepts the following arguments

const RooAbsPdf& model	The PDF to be studied
const RooArgSet& observables	The variables of the PDF to be considered the observables
FitModel(const RooAbsPdf&)	The PDF for fitting, if it is different from the PDF for generating
ConditionalObservables(const RooArgSet& set)	The set of observables that the PDF should *not* be normalized over
Binned(Bool_t flag)	Bin the dataset before fitting it. Speeds up fitting of large data samples
FitOptions(const char*)	Classic fit options, provided for backward compatibility
FitOptions(....)	Options to be used for fitting. All named arguments inside

Current status and plans

- RooFit is approaching ‘mature’ status
  - Most development involve tuning of interface and eliminating artifical (implementation-related) limitations
  - Most of the recent new features (such as named ranges) did not require major design changes
  - Used in many published physics analyses by BaBar (>4 year, >50 publications)
  - SourceForge download statistics suggest sizeable user community outside BaBar
- Concept of RooFit mostly revolves around user interface and p.d.f building
  - Not in the business of coding numeric integration methods, minimization packages, just want to interface them
- Code is now in ROOT5 as ‘external package’
  - ‘Large’ addition to ROOT (177 classes, 56K lines of code)
- Documentation upgrade main ongoing project at the moment.
  - ROOT-style ‘Users Guide’ (~100 pages)
  - Wiki interactive documentation for example, macros etc