The RooFit toolkit for data modeling

W. Verkerke
University of California Santa Barbara, Santa Barbara, CA 93106, USA
D. Kirkby
University of California Irvine, Irvine CA 92697, USA

`RooFit` is a library of C++ classes that facilitate data modeling in the ROOT environment. Mathematical concepts such as variables, (probability density) functions and integrals are represented as C++ objects. The package provides a flexible framework for building complex fit models through classes that mimic math operators, and is straightforward to extend. For all constructed models `RooFit` provides a concise yet powerful interface for fitting (binned and unbinned likelihood, χ^2), plotting and toy Monte Carlo generation as well as sophisticated tools to manage large scale projects. `RooFit` has matured into an industrial strength tool capable of running the BABAR experiment's most complicated fits and is now available to all users on SourceForge [1].

1. Introduction

One of the central challenges in performing a physics analysis is to accurately model the distributions of observable quantities $\vec{x}$ in terms of the physical parameters of interest $\vec{p}$ as well as other parameters $\vec{q}$ needed to describe detector effects such as resolution and efficiency. The resulting model consists of a "probability density function" (PDF) $F(\vec{x}\,;\vec{p},\vec{q})$ that is normalized over the allowed range of the observables $\vec{x}$ with respect to the parameters $\vec{p}$ and $\vec{q}$.

Experience in the BaBar experiment has demonstrated that the development of a suitable model, together with the tools needed to exploit it, is a frequent bottleneck of a physics analysis. For example, some analyses initially used binned fits to small samples to avoid the cost of developing an unbinned fit from scratch. To address this problem, a general-purpose toolkit for physics analysis modeling was started in 1999. This project fills a gap in the particle physicists' tool kit that had not previously been addressed.

A common observation is that once physicists are freed from the constraints of developing their model from scratch, they often use many observables simultaneously and introduce large numbers of parameters in order to optimally use the available data and available control samples.

2. Overview

The final stages of most particle physics analysis are performed in an interactive data analysis framework such as PAW[2] or ROOT[3]. These applications provide an interactive environment that is programmable via interpreted macros and have access to a graphical toolkit designed for visualization of particle physics data. The `RooFit` toolkit extends the ROOT analysis environment by providing, in addition to basics visualization and data processing tools, a language to describe data models.

The core features of `RooFit` are

- A natural and self-documenting vocabulary to build a model in terms of its building blocks (e.g., exponential decay, Argus function, Gaussian resolution) and how they are assembled (e.g., addition, composition, convolution). A template is provided for users to add new building-block PDFs specific to their problem domain.

- A data description language to specify the observable quantities being modeled using descriptive titles, units, and any cut ranges. Various data types are supported including real valued and discrete valued (e.g. decay mode). Data can be read from ASCII files or ROOT ntuples.

- Generic support for fitting any model to a dataset using a (weighted) unbinned or binned maximum likelihood, or χ^2 approach

- Tools for plotting data with correctly calculated errors, Poisson or binomial, and superimposing correctly normalized projections of a multidimensional model, or its components.

- Tools for creating a event samples from any model with Monte Carlo techniques, with some variables possibly taken from a prototype dataset, e.g. to more accurately model the statistical fluctuations in a particular sample.

- Computational efficiency. Models coded in `RooFit` should be as fast or faster than hand coded models. An array of automated optimization techniques is applied to any model without explicit need for user support.

- Bookkeeping tools for configuration management, automated PDF creation and automation of routine tasks such as goodness-of-fit studies.

3. Object-Oriented Mathematics

To keep the distance between a physicists' mathematical description of a data model and its implementation as small as possible, the `RooFit` interface is styled after the language of mathematics. The object-oriented ROOT environment is ideally suited for this approach: each mathematical object is represented by a C++ software object. Table I illustrates the correspondence between some basic mathematical concepts and `RooFit` classes.

Concept	Math Symbol	`RooFit` class name
Variable	x, p	`RooRealVar`
Function	$f(\vec{x})$	`RooAbsReal`
PDF	$F(\vec{x};\vec{p},\vec{q})$	`RooAbsPdf`
Space point	$\vec{x}$	`RooArgSet`
Integral	$\int_{\vec{x}_{min}}^{\vec{x}_{max}} f(\vec{x})d\vec{x}$	`RooRealIntegal`
List of space points	$\vec{x}_k$	`RooAbsData`

Table I Correspondence between mathematical concepts and `RooFit` classes.

Composite objects are built by creating all their components first. For example, a Gaussian probability density function with its variables

$$G(x,m,s) = \frac{\exp\left(-\frac{1}{2}\left(\frac{x-m}{s}\right)^2\right)}{\int_{x_L}^{x_H}\exp\left(-\frac{1}{2}\left(\frac{x-m}{s}\right)^2\right)}$$

is created as follows:

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

Each object has a name, the first argument, and a title, the second argument. The name serves as unique identifier of each object, the title can hold a more elaborate description of each object and only serves documentation purposes. All objects can be inspected with the universally implemented `Print()` method, which supports three verbosity levels. In its default terse mode, output is limited to one line, e.g.

```
root> f.Print() ;
RooRealVar::f:  0.531 +/- 0.072 L(0 - 1)
```

Object that represent variables, such as `RooRealVar` in the example above, store in addition to the value of that variable a series of associated properties, such as the validity range, a binning specification and their role in fits (constant vs. floating), which serve as default values in many situations.

Function objects are linked to their ingredients: the function object `g` *always* reflects the values of its input variables `x,m` and `s`. The absence of any explicit invocation of calculation methods allows for true symbolic manipulation in mathematical style.

`RooFit` implements its data models in terms of probability density functions, which are by definition positive definite and unit normalized:

$$\int_{\vec{x}_{min}}^{\vec{x}_{max}} f(\vec{x})d\vec{x} \ \equiv 1, \quad F(\vec{x},\vec{p}) \geq 0 \qquad (1)$$

One of the main benefits of probability density functions over regular functions is increased modularity: the interpretation of PDF parameters is independent of the context in which the PDF is used.

The normalization of probability density functions, traditionally one of the most difficult aspects to implement, is handled internally by `RooFit`: all PDF objects are automatically normalized to unity. If a specific PDF class doesn't provide its normalization internally, a variety of numerical techniques is used to calculate the normalization.

Composition of complex models from elementary PDFs is straightforward: a sum of two PDFs is a PDF, the product of two PDFs is a PDF. The `RooFit` toolkit provides as set of 'operator' PDF classes that represent the sum of any number of PDFs, the product of any number of PDFs and the convolution of two PDFs.

Existing PDF building blocks can be tailored using standard mathematical techniques: for example substituting a variable with a function in the preceding code fragment,

$$\begin{array}{l} m \rightarrow \ m(m_0,m_1,y) \ = m_0 + y * m_1 \\ \qquad\qquad\quad \downarrow \\ G(x, \ m(m_0,m_1,y) \ \ ,s) = G(x,y,m_0,m_1,s) \end{array}$$

is represented in exactly the same style in `RooFit` code:

```
RooRealVar m0("m0","mean offset",0) ;
RooRealVar m1("m1","mean slope",1) ;
RooRealVar y("y","y",0,3) ;
RooFormulaVar m("m","m0+y*m1",
                     RooArgList(m0,m1,y)) ;
RooGaussian g("g","gauss(x,m,s)",x,m,s) ;
```

Free-form interpreted C++ function and PDF objects such `RooFormulaVar` in the example above, are available to glue together larger building blocks. The universally applicable function composition operators and free-style interpreted functions make it possible to write probability density functions of arbitrary complexity in a straightforward mathematical form.

4. Composing and Using Data Models

We illustrate the process of building a model and its various uses with some example cases.

4.1. A One-Dimensional Yield Fit

The simplest and most common use of RooFit is to combine two or more library PDFs into a composite PDF to to determine the yield of signal and background events in a one-dimensional dataset.

The RooFit models library provides more than 20 basic probability density functions that are commonly used in high energy physics applications, including basics PDFs such Gaussian, exponential and polynomial shapes, physics inspired PDFs, e.g. decay functions, Breit-Wigner, Voigtian, ARGUS shape, Crystal Ball shape, and non-parametric PDFs (histogram and KEYS[4]).

In the example below we use two such PDFs: a Gaussian and an ARGUS background function:

```
// Observable
RooRealVar mes("mes","mass_ES",-10,10) ;

// Signal model and parameters
RooRealVar mB("mB","m(B0)",0) ;
RooRealVar w("w","Width of m(B0)",3) ;
RooGaussian G("G","G(meas,mB,width)",mes,mB,w) ;

// Background model and parameters
RooRealVar m0("m0","Beam energy / 2",-10,10) ;
RooRealVar k("k","ARGUS slope parameter",3) ;
RooArgusBG A("A","A(mes,m0,k)",mes,m0,k) ;

// Composite model and parameter
RooRealVar f("f","signal fraction",0,1) ;
RooAddPdf M("M","G+A",RooArgList(g,a),f) ;
```

The RooAddPdf operator class M combines the signal and background component PDFs with two parameters each into a composite PDF with five parameters:

$$\begin{aligned} M(m_{ES}; m_B, w, m_0, k, f) \;&=\; f \;\cdot\; G(m_{ES}; w, g) \\ &+\; (1-f) \;\cdot\; A(m_{ES}; m_0, k). \end{aligned}$$

Once the model M is constructed, a maximum likelihood fit can be performed with a single function call:

```
M.fitTo(*data) ;
```

Fits performed this way can be unbinned, binned and/or weighted, depending on the type of dataset provided[1]. The result of the fit, the new parameter values and their errors, are immediately reflected in the RooRealVar objects that represent the parameters of the PDF, mB,w,m0,k and f. Parameters can be fixed in a fit or bounded by modifying attributes of the parameter objects prior to the fit:

```
m0.setConstant(kTRUE) ;
f.setFitRange(0.5,0.9) ;
```

Visualization of the fit result is equally straightforward:

```
RooPlot* frame = mes.frame() ;
data->plotOn(frame) ;
M.plotOn(frame) ;
M.plotOn(frame,Components("A"),
                 LineStyle(kDashed)) ;
frame->Draw()
```

Figure 1 shows the result of the frame->Draw() operation in the above code fragment. A RooPlot object represents a one-dimensional view of a given observable. Attributes of the RooRealVar object mes provide default values for the properties of this view (range, binning, axis labels,...).

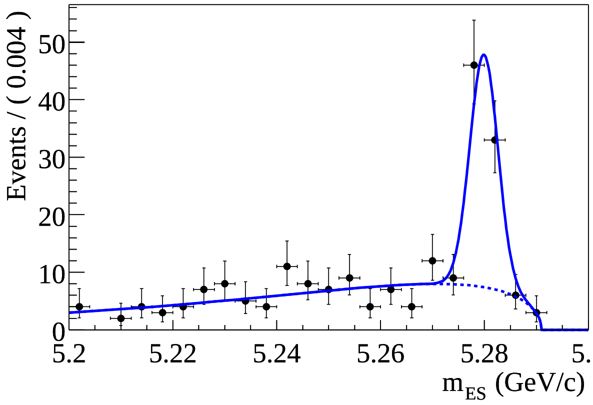

Figure 1: One dimensional plot with histogram of a dataset, overlaid by a projection of the PDF M. The histogram error are asymmetric, reflecting the Poisson confidence interval corresponding to a 1σ deviation. The PDF projection curve is automatically scaled to the size of the plotted dataset. The points that define the curve are chosen with an adaptive resolution-based technique that ensures a smooth appearance regardless of the dataset binning.

The plotOn() methods of datasets and functions accept optional arguments that modify the style and

[1] Binned data can be imported from a ROOT TH1/2/3 class, unbinned data can be imported from a ROOT TTree or a ASCII data file.

contents of what is drawn. The second `M.plotOn()` call in the preceding example illustrates some of the possibilities for functions: only the `A` component of the composite model `M` is drawn and the line style is changed to a dashed style. Additional options exists to modify the line color, width, filling, range, normalization and projection technique. The curve of the PDF is automatically normalized to the number of events of the dataset last plotted in the same frame. The points of the curve are chosen by an adaptive resolution-based technique: the deviation between the function value and the curve will not exceed a given tolerance[2] regardless of the binning of the plotted dataset.

The `plotOn()` method of datasets accepts options to change the binning, including non-uniform binning specifications, error calculation method and appearance. The default error bars drawn for a dataset are asymmetric and correspond to a Poisson confidence interval equivalent to 1σ for each bin content. A sum-of-weights error ($\sqrt{\Sigma_i w_i^2}$) can optionally be selected for use with weighted datasets. A special option, `Asym()`, is available to show asymmetry distributions of the type $\frac{N_A - N_B}{N_A + N_B}$. The errors bars will reflect a binomial confidence interval for such histograms.

4.2. A Simple Monte Carlo Study

`RooFit` PDFs universally support parameterized Monte Carlo event generation, e.g.

```
RooDataSet* mcdata = M.generate(mes,10000) ;
```

generates 10000 events in `mes` with the distribution of model `M`.

Events are by default generated with an accept/reject sampling method. PDF classes that are capable of generating events in a more efficient way, for example `RooGaussian`, can advertise and implement an internal generator method that will be used by `generate()` instead. Composite PDFs constructed with `RooAddPdf` delegate event generation to the component event generators for computational efficiency.

A common use of parameterized Monte Carlo is to study the stability and bias of a fit, in particular when when statistics are low[3]. A special tool is provided that automates the fitting and generating cycle for such studies and collects the relevant statistics. The example below examines the bias in the fraction parameter `fsig` of model `M`:

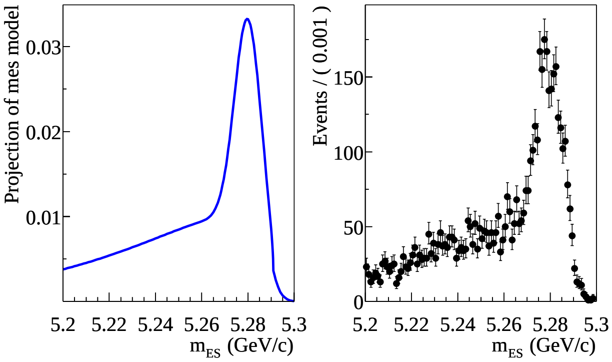

Figure 2: *Left:* Shape of PDF `M`. *Right:* Distribution of 10000 events generated from PDF `M`

```
// Generate and fit 1000 samples of 100 events
RooMCStudy mgr(M,M,mes) ;
mgr.generateAndFit(100,1000) ;

// Show distribution of fitted value of 'fsig'
RooPlot* frame1 = mgr.plotParam(fsig) ;

// Show pull distribution for 'fsig'
RooPlot* frame2 = mgr.plotPull(fsig) ;
```

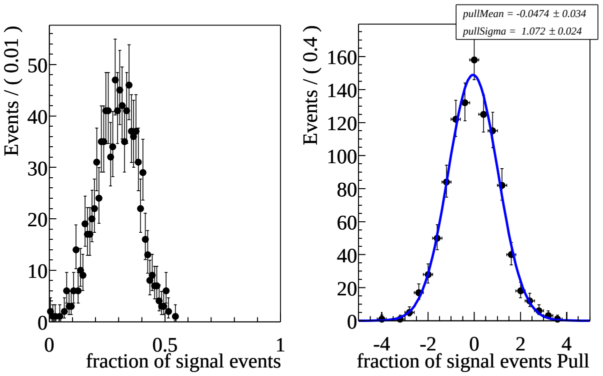

Figure 3: *Left:* distribution of fitted value of parameter `f` of model `M` to 1000 Monte Carlo data sets of 100 events each. *Right:* Corresponding pull distribution

4.3. Multi-Dimensional Models

Multi-dimensional data models are a natural extension of one-dimensional models in data analysis. Use of additional observables in the model enhances the statistical sensitivity of a fit, but are traditionally less frequently used due to additional logistical

[2] Tolerance is preset at 0.1% of the plot scale and recursively evaluated halfway between each adjacent pair of curve points

[3] (Likelihood) fits can exhibit an intrinsic bias that scale like $1/N$, where N is the number of events in the fitted dataset. At high statistics this bias is usually negligible compared to the statistical error, which scales like $1/\sqrt{N}$, but at low N the effect may be significant. See e.g. Eadie et al[5] for details.

and computational challenges. While *fitting* a multi-dimensional dataset is no more complex than fitting a one-dimensional dataset, the variety of ways in which the data and model can be visualized is much larger. In addition, the projection of a multi-dimensional model on a lower-dimensional view often requires non-trivial computations. RooFit greatly automates the visualization process and other aspects of multi-dimensional models so that use of multi-dimensional models is not substantially more complicated that use of one-dimensional models.

Multi-dimensional models can be constructed in a variety of ways:

- as a product of 1-dimensional PDFs
- as a fundamental multi-dimensional PDF object
- as a modified PDF that was originally intended for 1-dimensional use, e.g. $G(x; m, s) \to G(x; m(y, a, b), s) = G(x, y; a, b, s)$

We will illustrate some of the visualization options with a simple 3-dimensional model constructed as the product of three Gaussians summed with a product of three polynomials:

$$M = f \cdot G(x)G(y)G(z) + (1 - f) \cdot P(x)P(y)P(z)$$

encoded in RooFit classes as follows

```
RooRealVar x("x","x",-10,10) ;
RooRealVar mx("mx","mean x",0) ;
RooRealVar sx("sx","sigma x",3) ;
RooGaussian GX("GX","gauss(x,mx,sx)",x,mx,sx);
// Similar declarations of y,z, GY and GZ

RooRealVar ax("ax","bkg slope x",5) ;
RooPolynomial PX("PX","PX(x,ax)",x,ax) ;
// Similar declarations of PY and PZ

// Construct signal and background products
RooProdPdf S("S","sig",RooArgSet(GX,GY,GZ)) ;
RooProdPdf B("S","bkg",RooArgSet(PX,PY,PZ)) ;

// Construct sum of signal and background
RooRealVar fsig("fsig","signal frac.",0.,1.) ;
RooAddPdf M3("M3","S+B",RooArgList(S,B),fsig) ;
```

The RooProdPdf class represents the product of two or more PDFs. In most analysis applications such products factorize, e.g. $f(x) \cdot g(y) \cdot h(z)$, but non-factorizing products or partially factorizing products such as $f(x, y) \cdot g(y) \cdot h(z)$ are supported as well. Factorization of multi-dimensional PDFs greatly simplifies many expressions involved in fitting, plotting and event generation, and optimizations that take advantage of (partial) factorization properties are automatically applied.

4.3.1. Fitting

The procedure to fit the 3-dimensional model M3 to a 3-dimensional dataset data3 is identical to that of the 1-dimensional model M:

```
M3.fitTo(*data3) ;
```

4.3.2. Plotting

A three-dimensional model like M3 can be visualized in a number of ways. First there are the straightforward projections on the x,y and z axes:

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

RooPlot* yframe = y.frame() ;
data->plotOn(yframe) ;
model.plotOn(yframe) ;

RooPlot* zframe = z.frame() ;
data->plotOn(zframe) ;
model.plotOn(zframe) ;
```

While the invocation of the plotOn() method is identical to the one-dimensional case, the correspondence between the drawn curve and the PDF is more complicated: to match the projected distributions of multi-dimensional datasets, plotOn() must calculated a matching projection of the PDF. For a projection of a PDF $F(x, \vec{y})$ superimposed over the distribution of x of a dataset $D(x, \vec{y})$, this transformation is:

$$\mathcal{P}_F(x) = \frac{\int F(x, \vec{y}) d\vec{y}}{\int F(x, \vec{y}) dx d\vec{y}} \qquad (2)$$

For any dataset/PDF combination the set $\vec{y}$ of observables to be projected is automatically determined: each RooPlot object keeps track of the 'hidden' dimensions of each dataset and matches those to the dimensions of the PDF.

If the PDF F happens to be a factorizing product, like our signal and background PDFs S and B, Eq. 2 reduces to

$$\mathcal{P}_F(x) = \frac{F_x(x) \int F_{\vec{y}}(\vec{y}) d\vec{y}}{\int F_x(x) dx \int F_{\vec{y}}(\vec{y}) d\vec{y}} = \frac{F_x(x)}{\int F_x(x) dx} \qquad (3)$$

This particular optimization is automatically recognized and implemented by the RooProdPdf components of the M3 model. Non-factorizing multi-dimensional PDFs can also be drawn with RooAbsPdf::plotOn(), in such cases a combination of analytic and numeric integration techniques is used to calculate the projection.

In addition to the full projections on the observables x,y and z, it is often also desirable to view a projection of a *slice*, e.g. the projection on x in a narrow band of y, z or a box in y and z. Such projections are often desirable because they enhance the visibility of signal in the presence of a large background:

```
// Projection on X in a slice of Y
RooPlot* xframe = x.frame() ;
data->plotOn(xframe,"|y|<1") ;
model.plotOn(xframe,Slice(y,-1,+1)) ;


// Projection on Z in a slice of X and Y
RooPlot* zframe = z.frame() ;
data->plotOn(zframe,"|x|<1&&|y|<1") ;
model.plotOn(zframe,Slice(x,-1,+1)
                   ,Slice(y,-1,+1)) ;
```

While the Slice() option implements a (hyper)cubic slice of the data, RooFit also supports Monte Carlo projection techniques that allow to view regions of arbitrary shape. A common application of this technique is the 'likelihood projection plot', where a n-dimensional dataset and model are projected on one dimension after a cut on the likelihood of the model in the remaining $n-1$ dimensions. Figure 4 illustrates the power to enhance the signal visibility of such a projection. The likelihood projection technique naturally decomposes in a small number of RooFit operations: Figure 4 has been created with less than 10 lines of macro code.

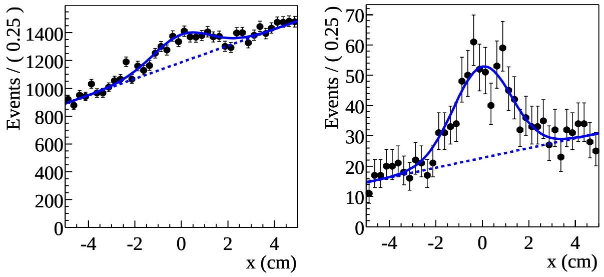

Figure 4: Example of a likelihood projection plot of model M3. *Left:* projection of full dataset and PDF on x. *Right:* Projection of dataset and PDF with a cut on the likelihood L_{yz}, calculated in the (y, z) projection of the PDF, at -5.0.

4.3.3. Event Generation

Event generation for multi-dimensional PDFs is similar to that for one-dimensional PDFs. This example

```
RooDataSet* mcdata =
        M3.generate(RooArgSet(x,y,z),10000) ;
```

generates a three-dimensional dataset of 10000 events.

If a model or its components are factorizable, such as the components S and B of model M3, the event generation will be performed separately for each factorizing group of dimensions. There is no limit to the number of observables in the event generator, but the presence of groups of two, three or more of observables that do not factorize will impact the performance of the accept/reject technique due to phase-space considerations.

Sometimes it is advantageous to not let generate all observables of the PDF, but take the distribution of some observables from an external dataset. This technique is commonly applied in fit validation studies where it is desirable to have a sample of Monte Carlo events that replicates the statistical fluctuations of the data as precisely as possible. RooFit facilitates this technique in the event generator with the concept of 'prototype datasets'. The prototype datasets prescribe the output of the event generator for a subset of the observables to be generated. To use the prototype feature of the generator, simply pass a dataset to the generate() method. For example, for a two-dimensional PDF M2, this code

```
RooDataSet* ydata ;
RooDataSet* mcdata = M2.generate(x,ydata) ;
```

generates a two-dimensional Monte Carlo event sample that has exactly the same distribution in y as the prototype dataset ydata, while the distribution of x is generated such that it follows the correlation between x and y that is defined by the PDF M2. Figure 5 illustrates the process: Fig. 5b shows the distribution of events generated the regular way from PDF M2 (5a). Fig. 5d shows the distribution in (x,y) when the distribution of events in y is taken from an external prototype dataset (5c): the distribution of events in y is exactly that of the prototype while the correlation between x and y encoded in the PDF is preserved.

4.4. Advanced Fitting Options

For fits that require more detailed control over the fitting process, or fits that require non-standard extensions of the goodness-of-fit quantity to be minimized, RooFit provides an alternate interface to the one-line fitTo() method. In this procedure the creation of the goodness-of-fit quantity to be minimized is separated from the minimum finding procedure.

RooFit provides two goodness-of-fit classes that represent the most commonly used fit scenarios

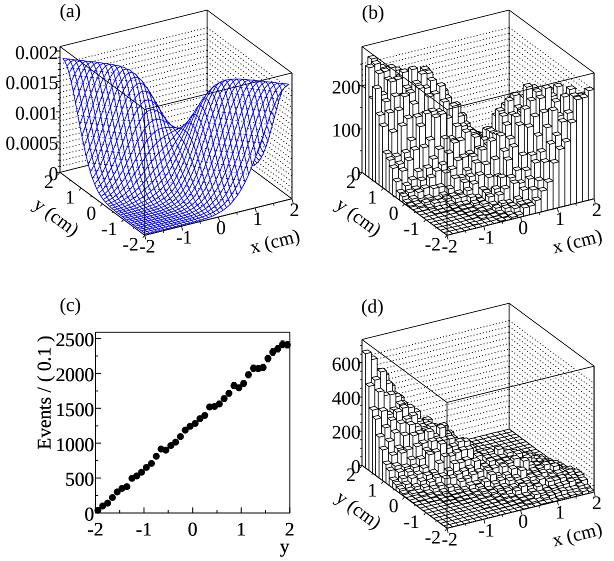

Figure 5: Demonstration of prototype-based Monte Carlo event generation. a) Two-dimensional PDF `M2`. b) Event sample generated from `M2`. c) One dimensional event sample in y with linear distribution. d) Event sample generated from `M2` using event sample shown in c) as prototype for y distribution.

- An (extended) negative log-likelihood implementation (`RooNLLVar`)

- A χ^2 implementation (`RooChi2Var`)

The χ^2 implementation uses by default the asymmetric Poisson errors for each bin of the dataset, but can be overridden to use (symmetric) sum-of-weights error ($\sqrt{\Sigma_i w_i^2}$) instead, for use with weighted datasets. Both classes represent their goodness-of-fit variable as a regular `RooFit` function so that all standard techniques can be applied. For example, plotting the likelihood defined by PDF `M` and dataset `data` as function of parameter `fsig` does not require any specialized plotting methods:

```
RooNLLVar nll("nll","-log(L)",M,data) ;
RooPlot* frame = fsig.frame() ;
nll.plotOn(frame) ;
```

Figure 6 shows the plot that results from the above code fragment.

The second step of the fit process, minimization of the goodness-of-fit quantity, is performed with MINUIT[6], via the interface class `RooMinuit`:

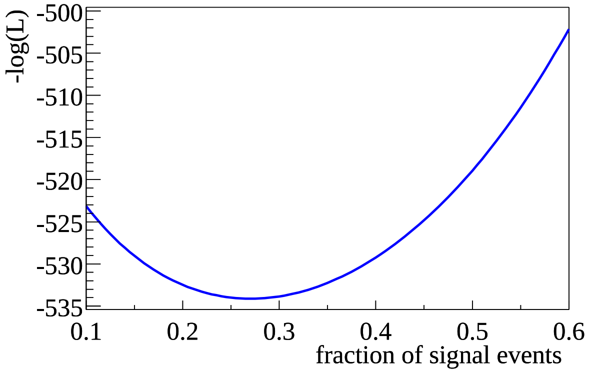

Figure 6: Negative log-likelihood defined by PDF `M` and dataset `data` as function of PDF parameter `fsig`

```
// Initialize a minimization session
RooMinuit m(nll) ;

// Invoke MIGRAD and HESSE
m.migrad() ;
m.hesse() ;

// Change value and status of some parameters
p1.setConstant() ;
p2.setConstant() ;
p2.setVal(0.3) ;

// Invoke MINOS
m.minos() ;

// Save a snapshot of the fitter status
RooFitResult* r = m.save() ;
```

In the above example, the MINUIT methods `migrad` and `hesse` are invoked first. The effect of each MINUIT operation is immediately reflected in the `RooRealVar` objects that represent the parameters of the fit model. Conversely, any changes to the fit parameter value or status are automatically propagated to MINUIT. The preceding code fragment illustrates how `RooFit` facilitates interactive fitting in C++. The `RooMinuit` interface almost completely isolates the user from any proprietary syntax usually needed to interact with fitters like MINUIT. One only needs to be familiar with the meaning of the basic MINUIT procedures: `migrad, hesse, minos, simplex, seek` and `contour`. The `save()` method saves a full snapshot of the MINUIT state: the initial and current parameter values, the global correlations, the full correlation matrix, status code and estimated distance to minimum.

Since the `RooMinuit` class can minimize *any* real-valued `RooFit` function, it is straightforward to fit customized goodness-of-fit expression. For example one can add a $\frac{1}{2}((p-\alpha)/\sigma_\alpha)^2$ penalty term to a standard negative log-likelihood using the RooFormulaVar class:

```
RooNLLVar nll(nll,nll,pdf,data) ;
RooFormulaVar nllp("nllp","penalized nll",
              "nll+0.5((p-alpha)/ealpha)^2",
              RooArgList(nll,p,alpha,ealpha)) ;
```

Similar modifications can be made to χ^2 definitions. It is also possible to develop an entirely different goodness-of-fit quantity by implementing a new class that derives from RooAbsOptGoodnessOfFit

5. Efficiency and Optimal Function Calculation

As the complexity of fits increases, efficient use of computing resources becomes increasingly important. To speed up the evaluation of probability density functions, optimization techniques such as value caching and factorized calculations can be used.

Traditionally such optimizations require a substantial programming effort due to the large amount of bookkeeping involved, and often result in incomplete use of available optimization techniques due to lack of time or expertise. Ultimately such optimizations represent a compromise between development cost, speed and flexibility.

RooFit radically changes this equation as the object-oriented structure of its PDFs allows centrally provided algorithms to analyze any PDFs structure and to apply generic optimization techniques to it. Examples of the various optimization techniques are

- *Precalculation of constant terms.*
  In a fit, parts of a PDF may depend exclusively on constant parameters. These components can be precalculated once and used throughout the fit session.

- *Caching and lazy evaluation.*
  Functions are only recalculated if any of their input has changed. The actual calculation is deferred to the moment that the function value is requested.

- *Factorization.*
  Objects representing a sum, product or convolution of other PDFs, can often be factorized from a single N-dimensional problem to a product of N easier-to-solve 1-dimensional problems.

- *Parallelization.*
  Calculation of likelihoods and other goodness-of-fit quantities can, due to their repetitive nature, easily be partitioned in to set of partial results that can be combined a posteriori. RooFit automates this process and can calculate partial results in separate processes, exploiting all available CPU power on multi-CPU hosts.

Optimizations are performed automatically prior to each potentially CPU intensive operation, and are tuned to the specific operation that follows. This realizes the maximum available optimization potential for every operation at no cost for the user.

6. Data and Project Management Tools

As analysis projects grow in complexity, users are often confronted with an increasing amount of logistical issues and bookkeeping tasks that may ultimately limit the complexity of their analysis. RooFit provides a variety of tools to ease the creation and management of large numbers of datasets and probability density functions such as

- *Discrete variables.*
  A discrete variable in RooFit is a variable with a finite set of named states. The naming of states, instead of enumerating them, facilitates symbolic notation and manipulation. Discrete variables can be used to consolidate multiple datasets into a single dataset, where the discrete variables states label the subset to which each events belongs.

- *Automated PDF building.*
  A common analysis technique is to classify the events of a dataset D into subsets D_i, and simultaneously fit a set of PDFs $P_i(\vec{x},\vec{p_i})$ to these subsets D_i. In cases were individually adjusted PDFs $P_i(\vec{x},\vec{p_i})$ can describe the data better than a single global PDF $P(\vec{x},\vec{p})$, a better statistical sensitivity can be obtained in the fit. Often, such PDFs do not differ in structure, just in the value of their parameters. RooFit offers a utility class to automate the creation the the PDFs $P_i(\vec{x},\vec{p_i})$: given a prototype PDF $P(\vec{x},\vec{p})$ and a set of rules that explain how the prototype should be altered for use in each subset (e.g. 'Each subset should have its own copy of parameter foo') this utility builds entire set of PDFs $P_i(\vec{x},\vec{p_i})$. It can handle an unlimited set of prototype PDFs, specialization rules and data subdivisions.

- *Project configuration management.*
  Advanced data analysis projects often need to store and retrieve projection configuration, such as initial parameters values, names of input files and other parameter that control the flow of execution. RooFit provides tools to store such information in a standardized way in easy-to-read ASCII files.

The use of standardized project management tools promotes structural similarity between analyses and increases users abilities to understand other RooFit projects and to exchange ideas and code.

7. Development Trajectory and Status

RooFit was initially released as RooFitTools in 1999 in the BaBar collaboration and started with a small subset of its present functionality. Extensive stress testing of the initial design by BaBar physicists in 1999 and 2000 revealed a great desire to have a tool like RooFit, but identified a number of bottlenecks and weaknesses, that could only be mitigated by a complete redesign.

The redesign effort was started in late 2000 and has introduced many of the core features that define RooFit in its current form: strong mathematical styling of the user interface, nearly complete absence of any implementation-level restrictions of any PDF building or utilization methods, efficient automated function optimization and powerful data and PDF management tools.

The new package, renamed RooFit, has been available to BaBar users since fall 2001. The functionality has been quite stable since early 2002 and most recent efforts have been spent on stabilization, fine tuning of the user interface and documentation. At present five tutorials comprising more than 250 slides and 20 educational macros are available, as well as a reference manual detailing the interface of all RooFit classes.

Since October 2002 RooFit is available to the entire HEP community: the code and documentation repository has been moved from BaBar to SourceForge, an OpenSource development platform, which provides easy and equal access to all HEP users. (http://roofit.sourceforge.net),

8. Current Use and Prospects

Since the package's initial release RooFit has been adopted by most BaBar physics analyses. Analysis topics include searches for rare B decays, measurements of B branching fractions and CP-violating rate asymmetries, time-dependent analyses of B and D decays to measure lifetime, mixing, and symmetry properties, and Dalitz analyses of B decays to determine form factors.

The enthusiastic adoption of RooFit within BaBar demonstrates the clear need and benefits of such tools in particle physics. Since its migration to SourceForge, RooFit is steadily gaining users from other HEP collaborations.

References

[1] http://roofit.sourceforge.net

[2] R. Brun et al., *Physics Analysis Workstation*, CERN Long Writeup Q121

[3] R. Brun and F Rademakers, *ROOT - An Object Oriented Data Analysis Framework*, Proceedings AIHENP'96 Workshop, Lausanne, Sep. 1996, Nucl. Inst. & Meth. in Phys. Res. A 389 (1997) 81-86. See also http://root.cern.ch

[4] K. Cranmer, *Kernel Estimation in High-Energy Physics*, Comp. Phys. Comm **136**, 198-207 (2001).

[5] Eadie et al, Statistical Methods in Experimental Physics, North Holland Publishing (1971)

[6] F. James, *MINUIT - Function Minimization and Error Analysis*, CERN Long Writeup D506