Fitting with RooFit

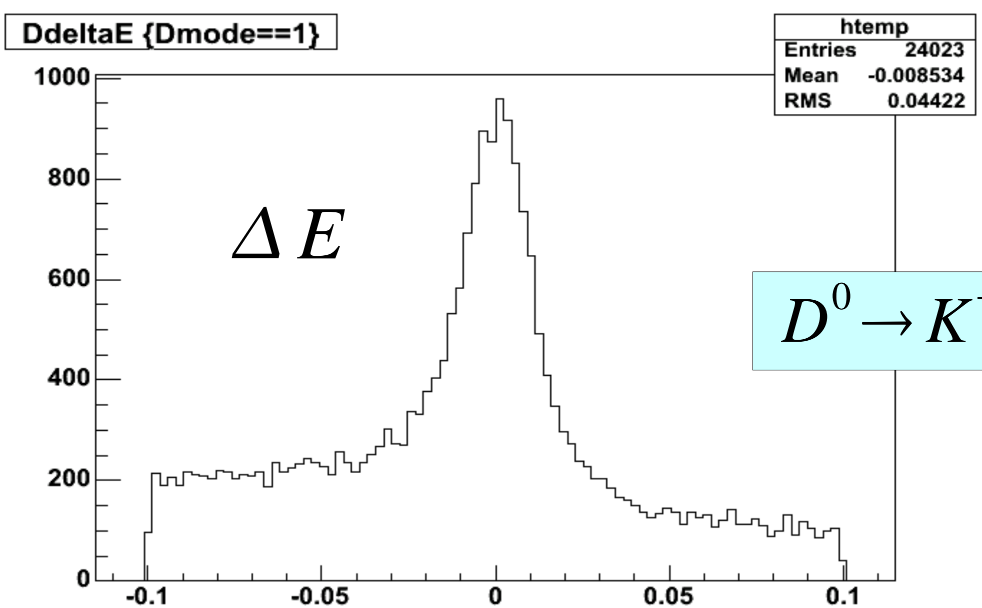
Steve Stroiney

CLEO 101 Day 8

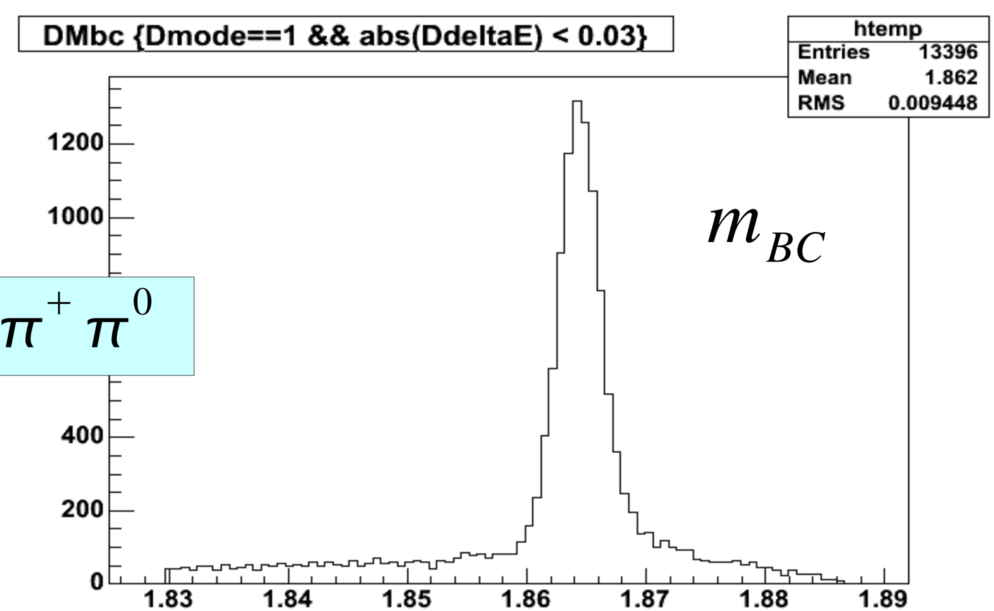
June 16, 2006

Why Do We Want to Fit?

- We can see from plots of ΔE and m_{BC}, for example, that CLEO produces D^0's, some of with decay to $K^- \pi^+ \pi^0$.
- But we also want to answer quantitative questions.
  - How many D^0's were observed?
  - What is the width ("resolution") of the ΔE and m_{BC} peaks?



$D^0 \rightarrow K^- \pi^+ \pi^0$



Fitting Terminology

- Our experiment measures a number of “observables” x_i such as ΔE and m_{BC}.
- We want to determine one or more “parameters” p_i from the data, such as the number of signal and background events, or the width of a peak.
- We describe the distribution of our observables by a “probability density function” (PDF) which is a function of both the observables and the parameters. We choose the PDF based on some guess about what function would match the data.

  typically one observable in a fit (1-D fit)

  – The PDF is normalized over the observables: $\int dx_1 \ldots dx_n f(x_1 \ldots x_n ; p_1 \ldots p_m) = 1$
  – $f(x, p)\, dx_1 \ldots dx_n$ is the probability that for a given event, the observables will be in the range $dx_1 \ldots dx_n$.
  – We vary the parameters to make the PDF match the actual distribution of the observables as well as possible. This is called “fitting.”

Unbinned Maximum Likelihood Fits

- You're probably familiar with a least-squares fit:
  - Bin data into a histogram, minimize $$\chi^2 = \sum_i \frac{(y_i - y_{expected})^2}{\sigma_i^2}$$
  - For a histogram, $y_i = N_i$ and $\sigma_i = \sqrt{N_i}$ (Poisson statistics in each bin)
- In RooFit, we'll use a fancier (and better) technique for fitting – an "unbinned maximum likelihood fit."
  - Likelihood $$L(p_1, \ldots p_m) = \prod_{\text{events } i} f(x_{1,i}, \ldots x_{n,i}; p_1, \ldots p_m)$$
  - Vary the parameters to maximize the likelihood. Equivalently, minimize
$$-\log[L(p_1, \ldots p_m)] = -\sum_{\text{events } i} \log[f(x_{1,i}, \ldots x_{n,i}; p_1, \ldots p_m)]$$
  - A least-squares fit mimics a maximum-likelihood fit if the bins are narrow and all bins have many events.
  - A maximum-likelihood fit is better (especially for small datasets), but it takes longer.

What is RooFit?

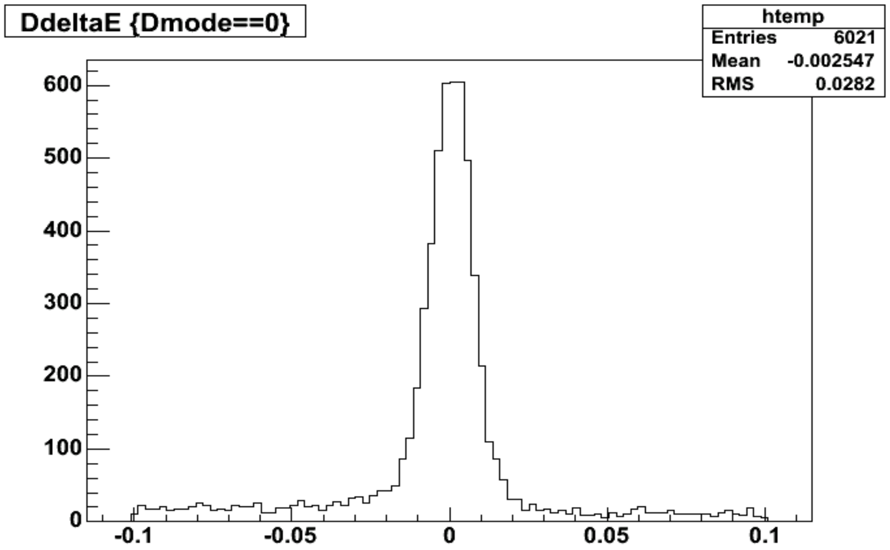
- ROOT is a collection of C++ classes.
  - ROOT can do fitting, but it isn't very convenient.
- RooFit provides more classes designed to make fitting easier.
  - We access RooFit through ROOT, and we can use any ROOT objects while using RooFit.
  - All RooFit class names begin with “Roo”.
  - RooFit handles many of the details for you, such as normalization.
- As of ROOT version 5, RooFit is included as part of ROOT. However, CLEO uses version 4, so we will have to load RooFit ourselves. (We will use version 1.92.)

Steps to Do a Fit

- Load RooFit into ROOT.
- Load events into a `RooDataSet`.
- Make cuts.
- Define PDF(s).
- Do the fit.
- Make a plot.

Ntuples Used in This Talk

- I'll be using ntuples created by RooFitExampleProc (available at www.lepp.cornell.edu/~srs63/private/cleo101_2006_day08/src/RooFitExampleProc)
  - This processor comes from `mkproc -dtag`, modified to create an ntuple with a few pieces of information for each DTag candidate.
- If you don't want to bother creating the ntuples yourself, you can get them at www.lepp.cornell.edu/~srs63/private/cleo101_2006_day08/ntuple/.
- In this talk, I'll show example code from fitting the ΔE distribution with a linear background and a Gaussian signal peak.
- The code outputs info for DTags in the following decay modes:
  - 0 $D^0 \rightarrow K^- \pi^+$
  - 1 $D^0 \rightarrow K^- \pi^+ \pi^0$
  - 200 $D^+ \rightarrow K^- \pi^+ \pi^+$
  - 201 $D^+ \rightarrow K^- \pi^+ \pi^+ \pi^0$



Loading RooFit Into ROOT

- Create a file called rootlogon.C in the directory from which you will run ROOT:

not necessary, and applies to ROOT generally

```
{
gROOT->SetStyle("Plain"); // remove gray background from plots
gSystem->Load("libMinuit.so") ;
gSystem->Load("/path/to/this/file/libRooFitCore.so") ;
gSystem->Load("/path/to/this/file/libRooFitModels.so") ;
using namespace RooFit ;
}
```

- In general, rootlogon.C is just a set of commands to be run every time ROOT starts.
- You can copy the file at
  www.lepp.cornell.edu/~srs63/private/cleo101_2006_day08/fits/rootlogon.C
- This file loads a copy of RooFit from
  `/nfs/cor/user/srs63/RooFit/V01-09-02_in_20060224_FULL_2/`
  This directory includes instructions on how to compile RooFit on you own, if you ever want to.
- You may get errors if you load RooFit in a different ROOT version.

Getting a RooDataSet

- We need to create a `RooDataSet` object to hold the events we want to fit and plot.
- A `RooDataSet` can be created in two different ways – from a text file or from a ROOT tree. I'll cover the second way.

name matches name in ROOT tree

```
// Create variables to load from the ntuple.
RooRealVar Dmode("Dmode", "decay mode of D candidate", -1, 1000);
RooRealVar DdeltaE("DdeltaE", "delta E of D candidate", -0.1, 0.1, "GeV");
RooArgSet ntupleVarSet(Dmode, DdeltaE);

TFile* file1 = new TFile("data33.root");
TTree* tree1 = file1->Get("RooFitExampleProc/nt1");
RooDataSet dataSet("ntuple_of_D_candidates", "ntuple of D candidates", tree1,
ntupleVarSet);

TFile* file2 = new TFile("data31.root");
TTree* tree2 = file2->Get("RooFitExampleProc/nt1");
RooDataSet tempDataSet("temp", "temporary data set", tree2, ntupleVarSet);
dataSet.append(tempDataSet);
```

only load events in this range

- See the file `headers/GetRooDataSet.c` on the Day 8 web page for a convenient function to create a `RooDataSet` from multiple files.
- Unfortunately, appending can be slow if the datasets are large.

Making Cuts

- We can make cuts on our data:

```
TCut DmodeCut = “Dmode == 0”; // Only accept DTags in decay mode 0 (D0 -> K-pi+)
RooDataSet* ReducedDataSet = (RooDataSet*)dataSet.reduce(DmodeCut);
```

- In order to cut on a certain variable, it must be in the `RooDataSet`. That's why we loaded `Dmode` into the dataset, but not the other variables.

Understanding RooRealVar

- All observables and parameters will be of type RooRealVar.
- Example: Variable constrained to be between -1 and 1:

```
RooRealVar myVar("myVar_name", "description of myVar", -1, 1, "units of myVar");
```

  - `"myVar_name"` is RooFit's internal identifier string for myVar.
  - The units argument is optional.
  - This form is typically used for RooRealVars that hold observables.
- Variable fixed to 0.5:

```
RooRealVar myFixedVar("myFixedVar_name", "description of myFixedVar", 0.5,
                       "units of myVar");
```

- Variable floating between -1 and 1, with initial value 0.5:

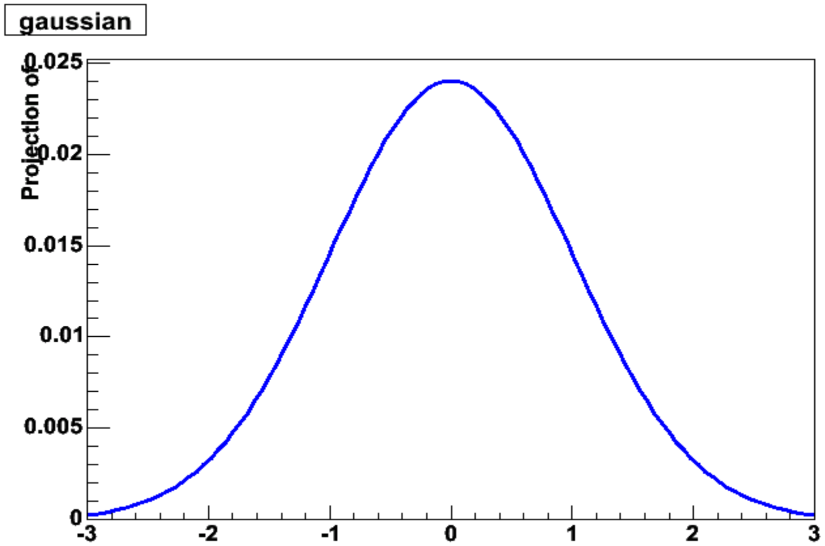
```
RooRealVar myFloatingVar("myFloatingVar_name", "description of myFloatingVar",
                          0.5, -1, 1, "units of myVar");
```

  - This form is typically used for fit parameters.

Defining a PDF: RooGaussian

- We'll use a Gaussian for the signal peak:

$$f(x)=\frac{1}{\sqrt{2\pi}\sigma}\exp\left\{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right\}$$



initial guess, lower bound, upper bound

```
// Define shape parameters
RooRealVar peak_gaussian_mean("peak_gaussian_mean", "mean of gaussian for signal
peak", 0, -0.1, 0.1, "GeV");
RooRealVar peak_gaussian_sigma("peak_gaussian_sigma", "width of gaussian for signal
peak", 0.01, 0.0, 0.05, "GeV");

// Define Gaussian PDF
RooGaussian peak_gaussian("peak_gaussian", "gaussian for signal peak", DdeltaE,
peak_gaussian_mean, peak_gaussian_sigma);

// Define yield parameter
RooRealVar peak_yield("peak_yield", "yield of signal peak", 1000, 0, 1000000);
```

Defining a PDF: RooPolynomial

- We'll use a linear function for the background.
- To create a linear function, we use the more general PDF RooPolynomial. For instance, for a cubic polynomial:

```
// Define shape parameters
RooRealVar poly_c1("poly_c1", "coefficient of x^1 term", 0, -10, 10);
RooRealVar poly_c2("poly_c2", "coefficient of x^2 term", 0, -10, 10);
RooRealVar poly_c3("poly_c3", "coefficient of x^3 term", 0, -10, 10);

// Define cubic polynomial PDF
RooPolynomial poly("poly", "cubic polynomial", x, RooArgList(poly_c1, poly_c2,
poly_c3) );
```

$$f(x) = 1 + c_1 x + c_2 x^2 + c_3 x^3$$

- For fitting the background of ΔE, a linear function:

```
// Define shape parameter
RooRealVar bkgd_poly_c1("bkgd_poly_c1", "slope of background", 0, -10, 10, "GeV^-
1");

// Define linear PDF
RooPolynomial bkgd_poly("bkgd_poly", "linear function for background", DdeltaE,
RooArgList(bkgd_poly_c1));
```

$$f(x) = 1 + c_1 x$$

- Incidentally, here's a constant (flat) function:

(no shape parameters)

```
RooPolynomial constant("constant", "flat function", x, RooArgList());
```

$$f(x) = 1$$

Creating the Total PDF

- Now, we need to add together our signal and background PDFs to create the total PDF that we'll fit to the ΔE distribution:

```
// Signal peak
RooRealVar peak_gaussian_mean("peak_gaussian_mean", "mean of gaussian for signal
peak", 0, -0.1, 0.1, "GeV");
RooRealVar peak_gaussian_sigma("peak_gaussian_sigma", "width of gaussian for signal
peak", 0.01, 0.0, 0.05, "GeV");
RooGaussian peak_gaussian("peak_gaussian", "gaussian for signal peak", DdeltaE,
peak_gaussian_mean, peak_gaussian_sigma);
RooRealVar peak_yield("peak_yield", "yield signal peak", 1000, 0, 1000000);

// Background
RooRealVar bkgd_poly_c1("bkgd_poly_c1", "slope of background", 0, -10, 10, "GeV^-
1");
RooPolynomial bkgd_poly("bkgd_poly", "linear function for background", DdeltaE,
RooArgList(bkgd_poly_c1));
RooRealVar bkgd_yield("bkgd_yield", "yield of background", 100, 0, 1000000);

// Total PDF
RooArgList shapes;
RooArgList yields;
shapes.add(bkgd_poly);      yields.add(bkgd_yield);
shapes.add(peak_gaussian);  yields.add(peak_yield);
RooAddPdf  totalPdf("totalPdf", "sum of signal and background PDF's", shapes,
yields);
```

A RooArgList is an ordered list of RooFit objects.

Doing the Fit

- Now, to do the fit, we just use:

```
totalPdf.fitTo(*ReducedDataSet, Extended() );
```

- `Extended()` tells RooFit to do an “extended likelihood” fit.
  - Instead of fixing the normalization to the number of events in the histogram (which is the default), an extended likelihood fit includes the overall yield as a parameter.
  - We need this type of fit if we are fitting for the yields of every component.
  - The uncertainties in the yields reported by the fit will include the statistical ($\sqrt{N}$) uncertainty.
  - If we weren't fitting for the yields, we wouldn't need this option.
- Other options are available to control and speed up the fit. They are listed in the RooFit user's guide on the RooFit web page, or at http://sourceforge.net/project/shownotes.php?release_id=308954 (search for “fitTo”)

More on Fitting

- RooFit uses MINUIT, a standard package, to do the fit.
- You will see MINUIT output on the screen. At the end, it should look like this:

```
FCN=-62473.3 FROM MINOS     STATUS=SUCCESSFUL    301 CALLS         492 TOTAL
                    EDM=0.000104002    STRATEGY= 1      ERROR MATRIX ACCURATE
  EXT PARAMETER                  PARABOLIC         MINOS ERRORS
  NO.   NAME      VALUE            ERROR      NEGATIVE      POSITIVE
   1  bkgd_poly_c1  -3.69964e+00   4.41224e-01  -4.37798e-01   4.44728e-01
   2  bkgd_yield   1.51699e+03   4.46900e+01  -4.42032e+01   4.51886e+01
   3  peak_gaussian_mean   4.38537e-04   1.15903e-04  -1.16119e-04   1.15696e-04
   4  peak_gaussian_sigma   7.00617e-03   1.00148e-04  -1.00171e-04   1.00124e-04
   5  peak_yield   4.50372e+03   7.05955e+01  -7.00657e+01   7.11417e+01
                               ERR DEF= 0.5
```

- If the fit is unsuccessful or if the error matrix is not accurate, you will have to tweak your fit functions and/or initial guesses for the parameters, and then redo the fit.
- If you include `Save()` as an argument to `fitTo()`, you will get a `RooFitResult` object containing more information, such as the covariance (aka error) matrix (which is sometimes very important!).

Plotting the Result

- OK, we've done the fit, and now we're ready to look at the result!

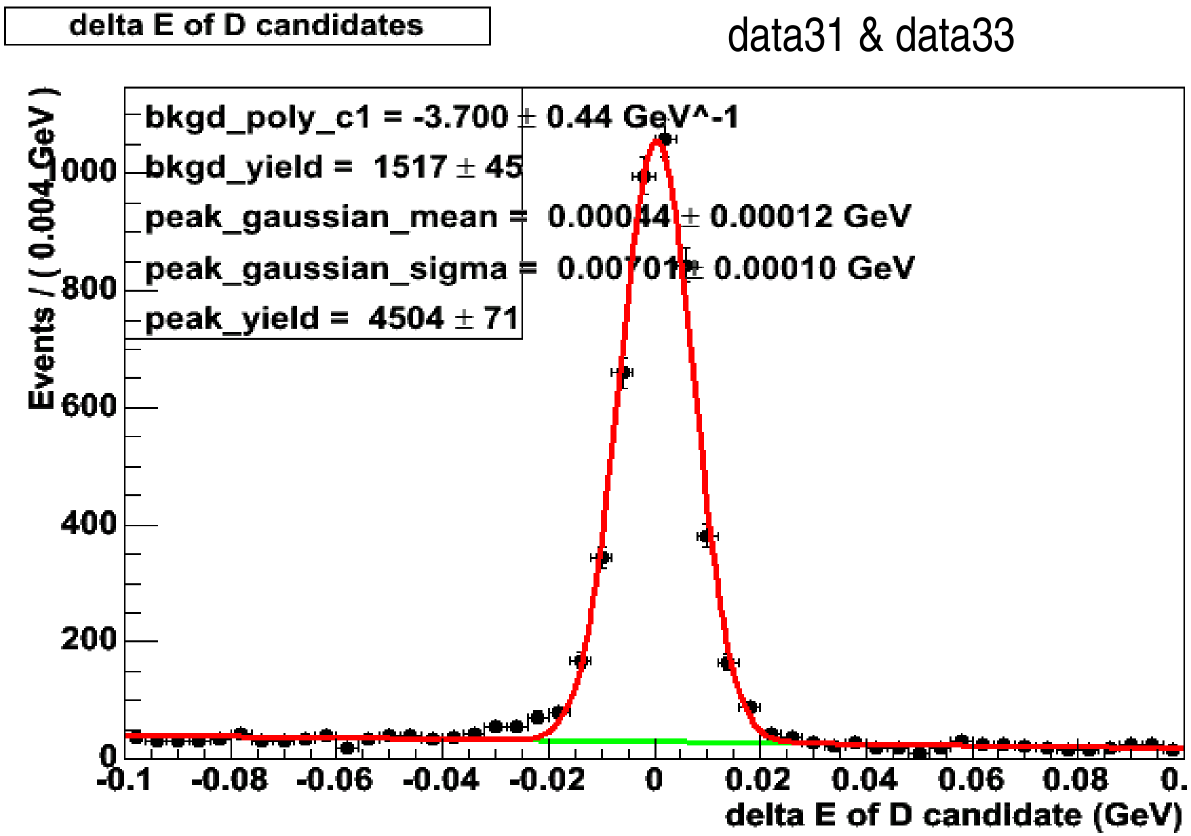
```
// Create RooPlot object with DdeltaE on the axis.
RooPlot* DdeltaEFrame=DdeltaE.frame(Bins(50), Name("deltaE"), Title("delta E of D
candidates") );

// Plot histogram of DdeltaE.
ReducedDataSet->plotOn(DdeltaEFrame, MarkerSize(0.9) );

// Display fit parameters.
totalPdf.paramOn(DdeltaEFrame, Format("NELU", AutoPrecision(2)), Layout(0.1, 0.4,
0.9) );

// Plot just the background.
totalPdf.plotOn( DdeltaEFrame, Components(bkgd_poly) , LineColor(kGreen) );

// Plot total PDF.
totalPdf.plotOn(DdeltaEFrame,LineColor(kRed)); // plot fit pdf

// The plot is not on the screen yet -- we have only created a RooPlot object.

// Put the plot on the screen.
TCanvas* DdeltaEcanvas = new TCanvas("DdeltaEcanvas", "Fit of delta E"); // make
new canvas
DdeltaEFrame->Draw(); // Put our plot on the canvas.
```

A Look at the Result

- Here's what we see on the screen. The green line is the background, and the red line is the sum of the background and signal PDF's.
- Notice the peak and signal yields. The uncertainties are larger than $\sqrt{N}$.
- The fit looks good, except around -0.02. This excess is due to initial state radiation (when the initial state radiates a photon before forming $c\bar{c}$).

$D^0 \to K^- \pi^+$ (mode 0)

data31 & data33

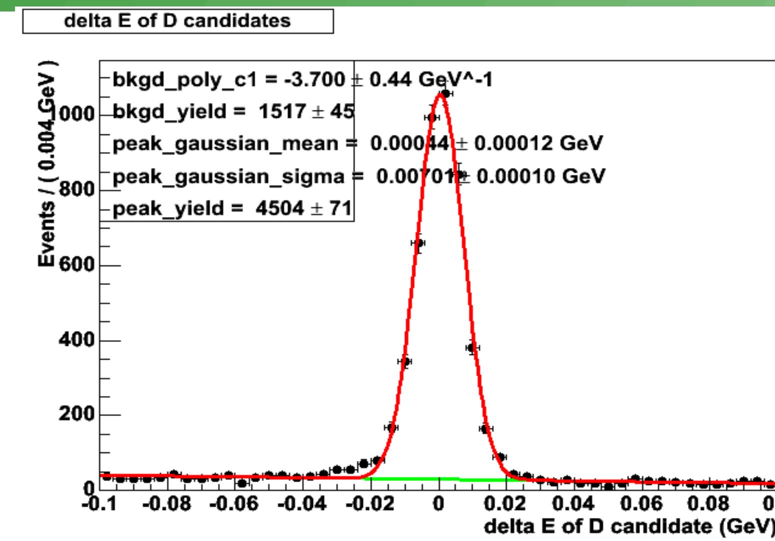


Additional Topics

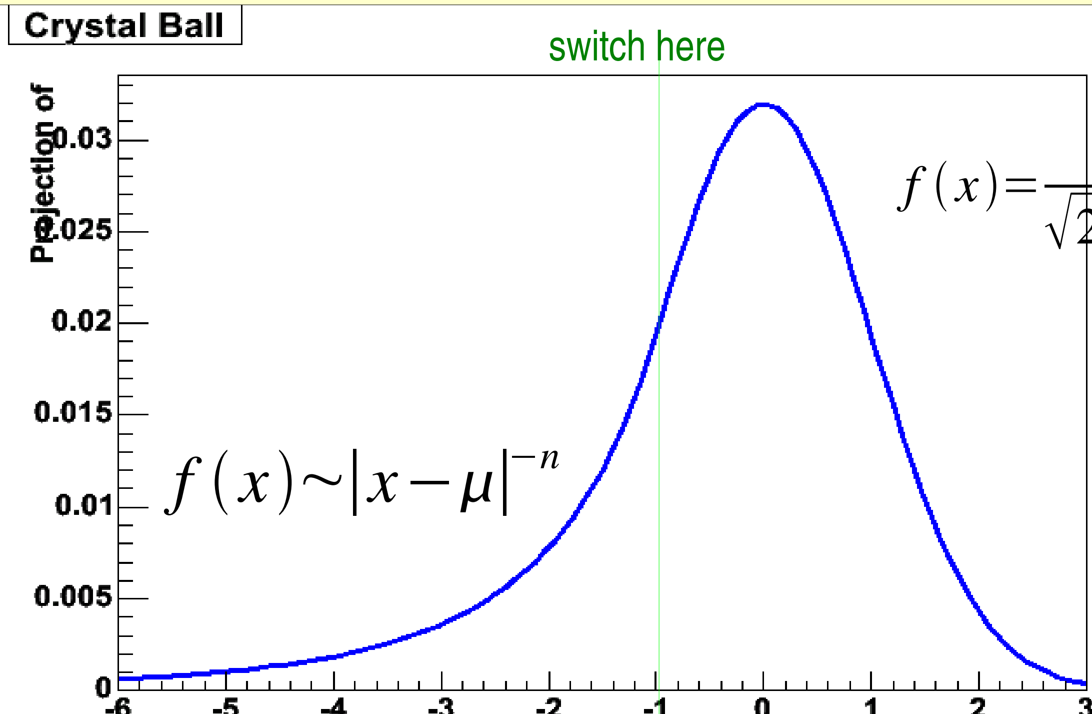
- OK, now we've seen the basics of how to do a fit with RooFit.
- Let's look at a few more things that will be useful:
  - More fitting functions
    - RooCBShape
    - RooArgusBG
    - RooGenericPdf
    - RooAddPdf (another form)
  - RooFormulaVar
  - Doing integrals

Fitting Functions: RooCBShape

- What do we do when we have a peak that looks mostly like a Gaussian, but with a tail on one side?
- Use a “Crystal Ball” shape.
  - Gaussian with power-law tail.




```
RooCBShape CBall(“CBall”, “Crystal Ball shape”, x, mean, sigma, alpha, n);
```



$$f(x) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left\{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right\}$$

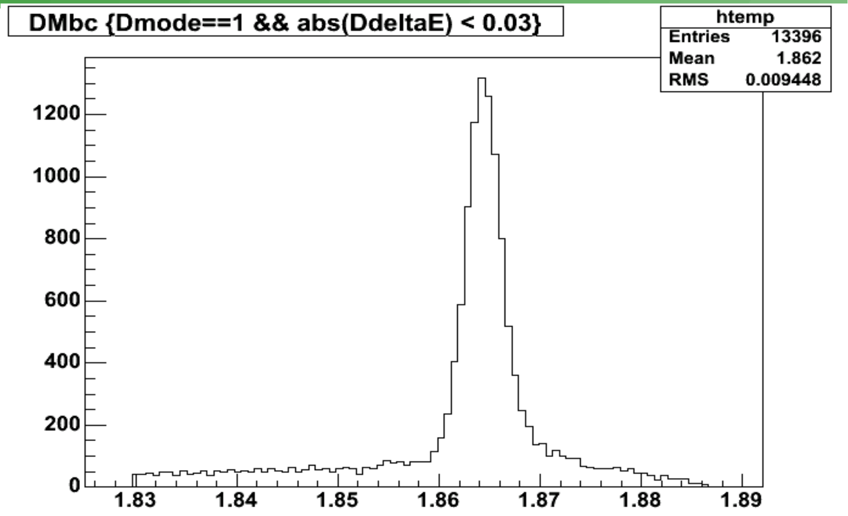
$$f(x) \sim |x-\mu|^{-n}$$

$$x - \sigma\alpha$$

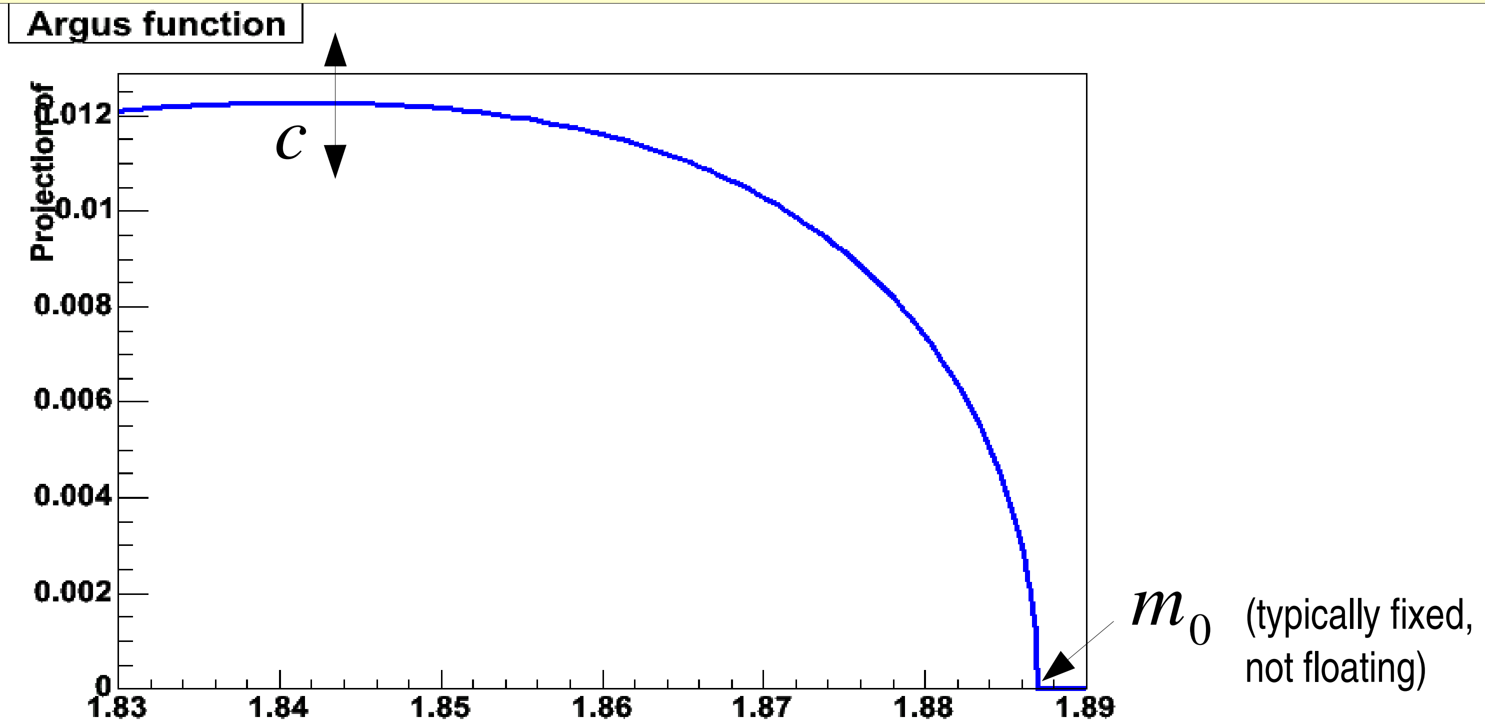
For a tail on the right side, use a negative alpha.

Fitting Functions: RooArgusBG

- What function can we use for the combinatoric background in m_{BC}?
- Use an "Argus" shape.
  - Cutoff (m_0) at the beam energy.




```
RooArgusBG Argus("Argus", "Argus shape", x, m0, c);
```



Fitting Functions: RooGenericPdf

- If you want to use a function that RooFit doesn't provide, you can use a RooGenericPdf. For example:

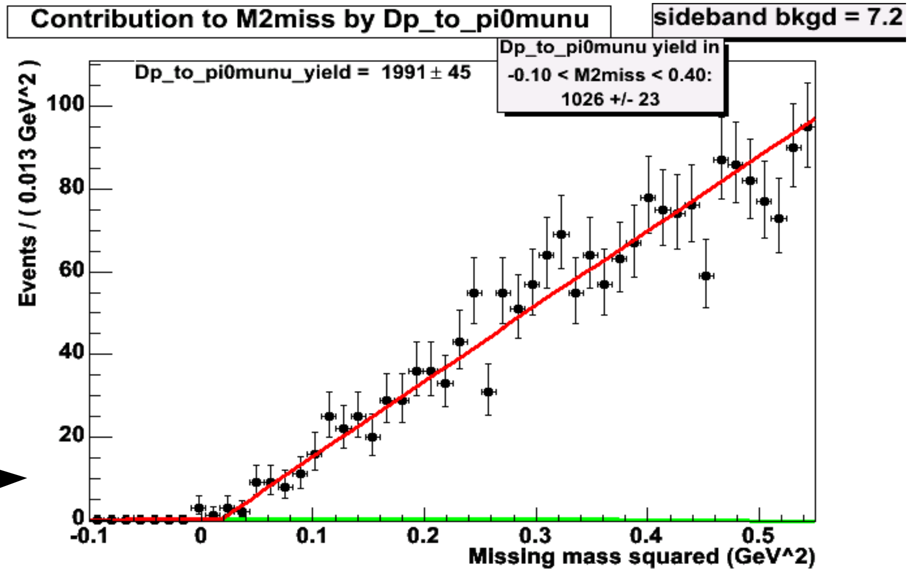
```
RooRealVar cutOffVar("cutOff", "", 0.0182);
RooGenericPdf nonNegLine("nonNegLine", "", "(xVar > cutOff)*(xVar – cutOff)",
                         RooArgList(xVar, cutOffVar) );
```

- Equivalently,

```
RooRealVar cutOffVar("cutOff", "", 0.0182);
RooGenericPdf nonNegLine("nonNegLine", "", "(@0 > @1)*(@0 - @1)",
                         RooArgList(xVar, cutOffVar) );
```

  – `@0` refers to the first element of the `RooArgList`, `@1` refers to the second, `@2` to the third, etc.

- Notice that RooFit handles the normalization.
- Here's what it looks like:



RooAddPdf Explained More

- We already saw how to use `RooAddPdf` to get the sum of two PDF's. In fact, we can add any number of PDF's in this way:

```
RooAddPdf sumWithYields("sumWithYields", "", RooArgList(func1, func2, func3),
                        RooArgList(yield1, yield2, yield3) );
```

- Here, `func[1-3]` are any three PDF's, and `yield[1-3]` are the number of events in each of them.

- Another way to use `RooAddPdf` is to specify a sum of n functions with n-1 parameters for the fraction of the total found in each function:

```
RooAddPdf sumWithFracs("sumWithFracs", "", RooArgList(func1, func2, func3),
                       RooArgList(frac1, frac2) );
```

- Here, `frac1` is the fraction of events in `sum2` that are found in `func1`, and `frac2` is the fraction in `func2`. There is no `frac3` since it's equal to `1-frac1-frac2`.

- We can then specify a yield for `sumWithFracs`, just like any normal PDF.

RooFormulaVar

- If you want to define one fit parameter in terms of other fit parameters, use a `RooFormulaVar`.
- For example,

```
RooRealVar var1("var1", "", 2, 1, 3);
RooRealVar var2OverVar1("var2OverVar1", "", 2, 0, 999);
RooFormulaVar var2("var2","", "var2OverVar1*Var1", RooArgList(var2OverVar1, var1));
```

or "@0*@1", like in `RooGenericPdf`

- `var2` can be used anywhere a `RooRealVar` can be used. But then the fit will use `var1` and `var2OverVar1` as the fit parameters.
- This can also be useful if you want to make two variables "float together." For instance, we could make `var2OverVar1` be constant.

Two Gaussians Example

- For example, suppose we want to create a function which is the sum of two Gaussians with the same mean but different widths:

```
RooRealVar mean("mean", "", 0, -3, 3);
RooRealVar width1("width1", "", 2, 1, 3);
RooRealVar width2OverWidth1("width2OverWidth1", "", 2, 0, 999);
RooFormulaVar width2("width2","", "@0*@1", RooArgList(width2OverWidth1, width1));
RooRealVar frac1("frac1", "fraction of events in gauss1", 0.5, 0, 1);

RooGaussian gauss1("gauss1", "first Gaussian",  xVar, mean, width1);
RooGaussian gauss2("gauss2", "second Gaussian", xVar, mean, width2);
RooAddPdf twoGaussians("twoGaussians", "sum of two Gaussians",
                        RooArgList(gauss1, gauss2), RooArgList(frac1));
```

- If you wanted to, you could make the ratio of widths fixed by setting `width2OverWidth1` to be constant instead of floating:

```
RooRealVar width2OverWidth1("width2OverWidth1", "", 2);
```

Doing Integrals

- The fit gives us the yield of each component of the histogram, but what if we want to know the yield in a certain range of the histogram?
- We can find out the fraction of a PDF within a certain region with the following function, available at the Day 8 page at `headers/NormalizedIntegral.c`

```
double NormalizedIntegral(RooAbsPdf * function, RooRealVar& integrationVar,
                            double lowerLimit, double upperLimit)
{
    integrationVar.setRange("integralRange", lowerLimit, upperLimit) ;
    RooAbsReal* integral = (*function).createIntegral(integrationVar,
                        NormSet(integrationVar), Range("integralRange")) ;
    double normalizedIntegralValue = integral->getVal();

    printf("\nIntegrating %s from %f to %f\n", (*function).GetName(),
                                                lowerLimit, upperLimit);
    printf("Integral value: %f\n", normalizedIntegralValue);

    return normalizedIntegralValue;
}
```

Advanced Topics

- Here are some advanced things you can do with RooFit that aren't covered in this talk:
  - Plot on a log scale. (See `fits/fitDeltaE.cc` on the Day 8 web page.)
  - Output `RooRealVars` to a text file, and load them from a text file.
  - Write your own fit function. (For instance, an error function – the integral of a Gaussian – called `RooErf`.)
  - Do two-dimensional fits.
  - Use "blind" parameters.
  - Load data from a text file. (In short, use
    `RooDataSet::read(fileName, variable)`
    where `fileName` is a file with one number per line, and `variable` is a `RooRealVar`.)

Additional Resources

- RooFit web page
  - http://roofit.sourceforge.net
  - This page contains a user's manual and some (old) tutorial slides. Also, you can look up all the RooFit classes and functions.
- ROOT web page (since you're also using ROOT classes)
  - http://root.cern.ch/
- Some (maybe) useful utilities I've written in the headers area of the day 8 web page www.lepp.cornell.edu/~srs63/private/cleo101_2006_day08/fits/headers/
- Me, or other RooFit users

Things to Try

- If you want to, run RooFitExampleProc over a dataset or two. It generates an ntuple with very basic information about DTags in 4 decay modes. If you prefer, you can use the ntuples on the day 8 web page (so you don't have to bother with processors or suez at all!).
- Run the function in fitDeltaE.cc (available on the day 9 page). To run it:

```
.L fitDeltaE.cc
fitDeltaE(0) // for D0->Kpi (1 for Kpipi0, 200 for D+ -> Kpipi, 201 for Kpipipi0)
```

- Examine the code in fitDeltaE.cc to see what it does. Uncomment some of the lines I've commented out, and observe the results.
- Modify fitDeltaE.cc to use a Crystal Ball instead of a Gaussian for the signal peak. (This will help with the tail on the left side of the peak.)
- Try to fit the Mbc distribution with an Argus background shape and a peak of some sort (Gaussian, Crystal Ball, Gaussian + Crystal Ball). Much of the code can be copied from fitDeltaE.cc – just switch deltaE to Mbc and change the fit functions. I've written a function in fitMbc.cc to do this, so look at that if you get stuck. However, you should try to write it yourself first. (You may want to look at a plot of Mbc first before you try to fit it.)

Everything Begins with “Roo”!

The End