AFit *Utilities*

http://pprc.qmul.ac.uk/~bevan/afit/

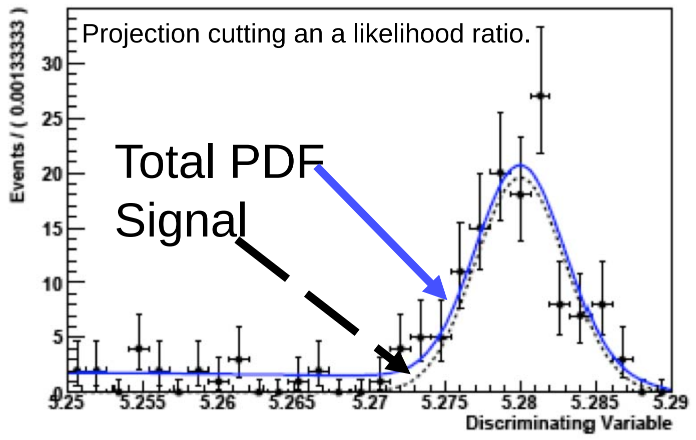
Adrian Bevan

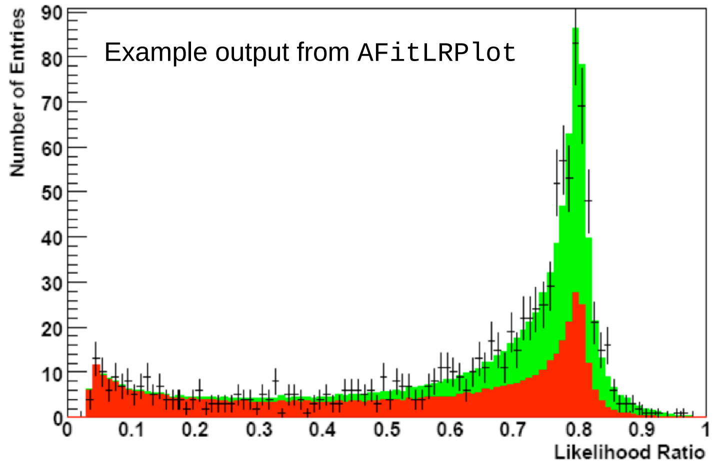
Overview

- AFit Utility overview.
- Checking correlations between variables.
- Interface to TMVA.
- Running ensembles of toy Monte Carlo experiments.
- Plotting:
- Summary

Utilities

- When setting up your analysis you will think about
  - Checking for correlations between fit variables: `AFitStatTools`
  - Defining an MVA: `AFitTMVAInterface`
  - Running toys: `AFitToy`
  - Plotting: `AFitProjectionPlot`
    - Projections, (not)cutting on data, on likelihood ratio: S/(S+B)
  - Likelihood ratio plot to test global agreement between fit and MC: `AFitLRPlot`





Checking Correlations

- The problem:
  - A likelihood fit usually assumes that discriminating variables are uncorrelated.
  - Thus: $$\mathcal{P}(\underline{x}) = \prod_i \mathcal{P}(x_i)$$
  - This is often an approximation.
  - Several ways to deal with this:
    - i) Transform the basis vector to remove correlations:
    - ii) Check to see if the correlations matter:
      - A) Compute correlation matrix of the discriminating variables

        [If ‘small’ then can probably neglect correlations].
      - B) Run ensembles of embedded toy Monte Carlo, using fully simulated data where possible to determine systematic uncertainties.

Checking Correlations

- The AFitStatTools class provides a simple interface for you to compute correlations between pairs of variables in a tree.

```
AFitStatTools st;
st.correlation(myTree, "x,y,z", kFALSE);
```

A tree containing the variables you want to compare.

A comma separated list of variables.

kFLASE: this is the default value and means that Pearson correlation coefficients are computed.

- The output of this command for a set of 500 randomly generated x,y,z values will look something like the following:

```
Results from AFitStatTools::pearsons_correlation for the variables
    x,y,z
Pearsons Correlation matrix follows:
x      1        -0.0862659    -0.0524787
y  -0.0862659        1         0.0436937
z  -0.0524787    0.0436937             1
       x            y             z
```

Each entry in this matrix is computed using:

$$\rho_{i,j} = \frac{\sigma_{i,j}}{\sigma_i \sigma_j}$$

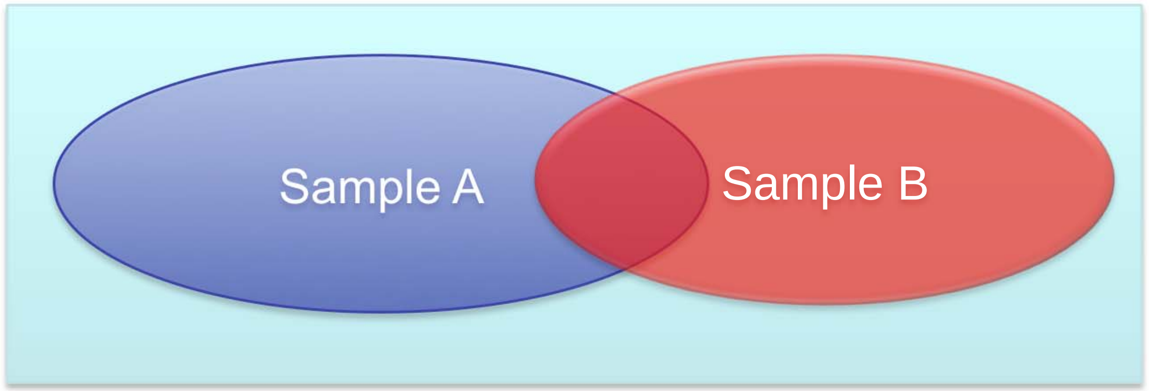
where σ_i is the RMS of the i distribution, and $\sigma_{i,j}$ is the covariance.

TMVA Interface

- TMVA is a toolkit for multivariate analysis described in detail on the TMVA homepage:

  http://tmva.sourceforge.net/

- The `AFitTMVAInterface` class is a wrapper to TMVA.
- See the TMVA user guide to learn how to use this package, as only the `AFitTMVAInterface` will be described here.
- The Problem:



How does one distinguish between two samples A and B when they overlap in a non-trivial way?

TMVA Interface

- Given two samples of data: A and B (e.g. signal and background) there are many ways to classify them.
  - Each classifier can be used to assign a probability of the A*ness* or B*ness* of an event.
  - Classifiers implemented:

Cuts
Likelihood
HMatrix
Fisher
MLP
CFMlpANN
TMlpANN
BDT
RuleFit
SVM
BayesClassifier
Committee
MaxMethod

Step 1: Write a configuration file specifying:
- Signal data
- Background data
- Output file
- Classifiers to use (`trainingMethod`)
- Variables to use
- TMVAFactory options
- Training options

Step 2: `trainMethods` to run through all classifier training steps, and output results.

Step 3: `runReader` to add the classifier values to a RooDataSet.

TMVA Interface

- Example configuration file:

```
[TMVAInterface]
sigFile = tmva_fsig.root
bgFile  = tmva_fbg.root
dataName = data
outputFileName = tmva_out.root
trainingMethod = Fisher,CFMlpANN,TMlpANN,BDT,RuleFit,SVM,MLP
variables = a:F,b:F,c:F

factoryOptions = V,Color
trainingOptions = NSigTrain=500:NBkgTrain=500:NSigTest=500:NbkgTest=500:SplitMode=Random
```

Specifying the input and output files

Comma separated list of classifiers

Variables to use

- Once the configuration file is ready, it is simple to run TMVA:

```
AFitTMVAInterface a("myTMVAConfigFile.txt");
a.trainMethods();
```

- ... and straightforward to add all classifiers to a data set:

```
a.runReader("tmva_fsig.root", "tmva_outdata.root");
```

Toy MC

- The problems:
  - Likelihood functions are intrinsically biased.
    - Need to run ensembles of PDF toy Monte Carlo experiments.
    - Need to verify that the likelihood you define as an acceptable, or negligible bias with regard to extracting your observables.
    - Look at pull distributions:

$$Pull(x) = \frac{\left(x_{obs} - x_{input}\right)}{\sigma\left(x\right)}$$

    - Rule of thumb (valid for large statistics):
      - Pull mean ~0 (if fit is unbiased)
      - Pull width ~1 (if errors are estimated correctly)
  - Correlations between discriminating variables:
    - Need to run ensembles of embedded toy Monte Carlo experiments.

Toy MC

- The problems:
  - Likelihood functions are intrinsically biased.
    - Need to run ensembles of PDF toy Monte Carlo experiments.
    - Need to verify that the likelihood you define as an acceptable, or negligible bias with regard to extracting your observables.
    - Look at pull distributions:

$$Pull(x) = \frac{\left(x_{obs} - x_{input}\right)}{\sigma\left(x\right)}$$

Fitted value of the parameter.

Parameter value used in generation (from datacard)

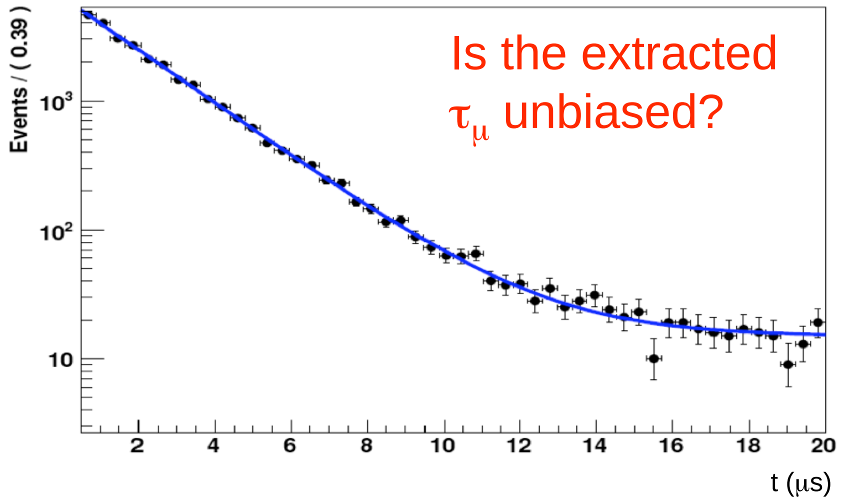
Error on x extracted from fit.

    - Rule of thumb (valid for large statistics):
      - Pull mean ~0 (if fit is unbiased)
      - Pull width ~1 (if errors are estimated correctly)
  - Correlations between discriminating variables:
    - Need to run ensembles of embedded toy Monte Carlo experiments.

Toy MC [PDF Toys]

- Using the τ_μ example (see the user guide, web, and examples)
  - Fit the time distribution with an exponential signal + constant background model.
  - Extract signal and background yields, as well as τ_μ.

```
AFitMaster master("MuonLifetime.txt");
RooAbsPdf * pdf = master.getPdf();

RooArgSet * compSet = pdf->getComponents();
RooArgSet * parSet  = pdf->getParameters(compSet);
parSet.Print("v");

RooRealVar * t = (RooRealVar*)parSet->find("t");

AFitToy toy;
toy.setOutputDir("toy/");

for(int i=0; i < 10; i++){
    toy.setSeed(i);
    // reset parameters to intial values

    RooDataSet * data = toy.generateToySample(pdf, RooArgSet(*t), 1000, 0);

     // do something … fit the data or save the file to disk
}
```



Is the extracted τ_μ unbiased?

Toy MC [PDF Toys]

- Using the τ_μ example (see the user guide, web, and examples)

Build the fit model in the normal way.

```
AFitMaster master("MuonLifetime.txt");
RooAbsPdf * pdf = master.getPdf();

RooArgSet * compSet = pdf->getComponents();
RooArgSet * parSet  = pdf->getParameters(compSet);
parSet.Print("v");

RooRealVar * t = (RooRealVar*)parSet->find("t");

AFitToy toy;
toy.setOutputDir("toy/");

for(int i=0; i < 10; i++){
   toy.setSeed(i);
   // reset parameters to intial values

   RooDataSet * data = toy.generateToySample(pdf, RooArgSet(*t), 1000, 0);

    // do something … fit the data or save the file to disk
}
```

Determine the discriminating variables to generate (just “t” in this case).

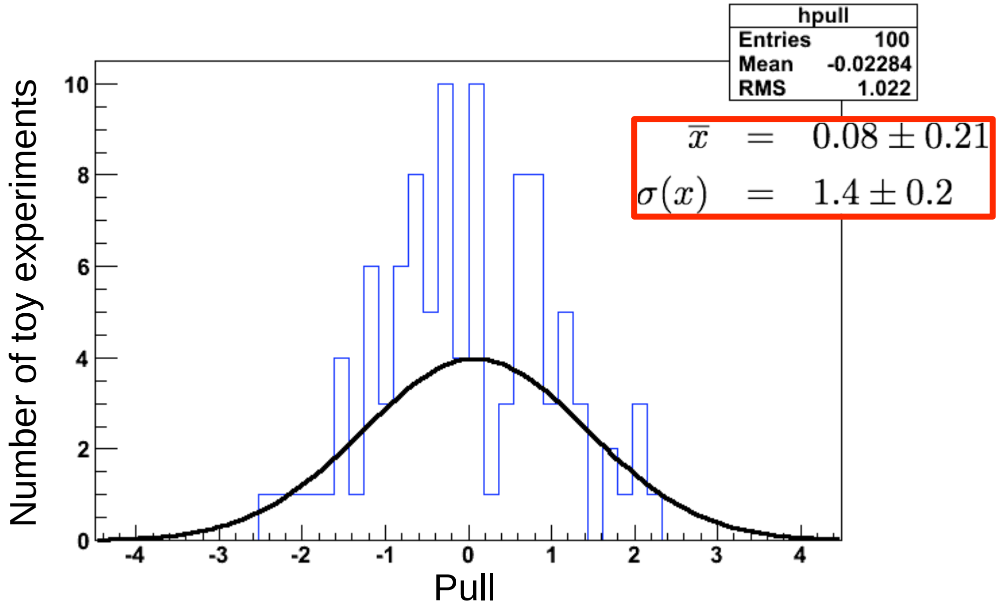
Configure an AFitToy object

Generate (and fit) toy data from your fit model.

Toy MC [PDF Toys]

- Is the fit behaving sensibly?
  - Q) Are we able to fit back the value of τ_{μ} we put into the toy?
  - Q) Is the error we get back from MINUIT sensible?

- Remember that we expect a Gaussian distribution with mean and width of the pull distribution equal to 1 and 0, respectively.



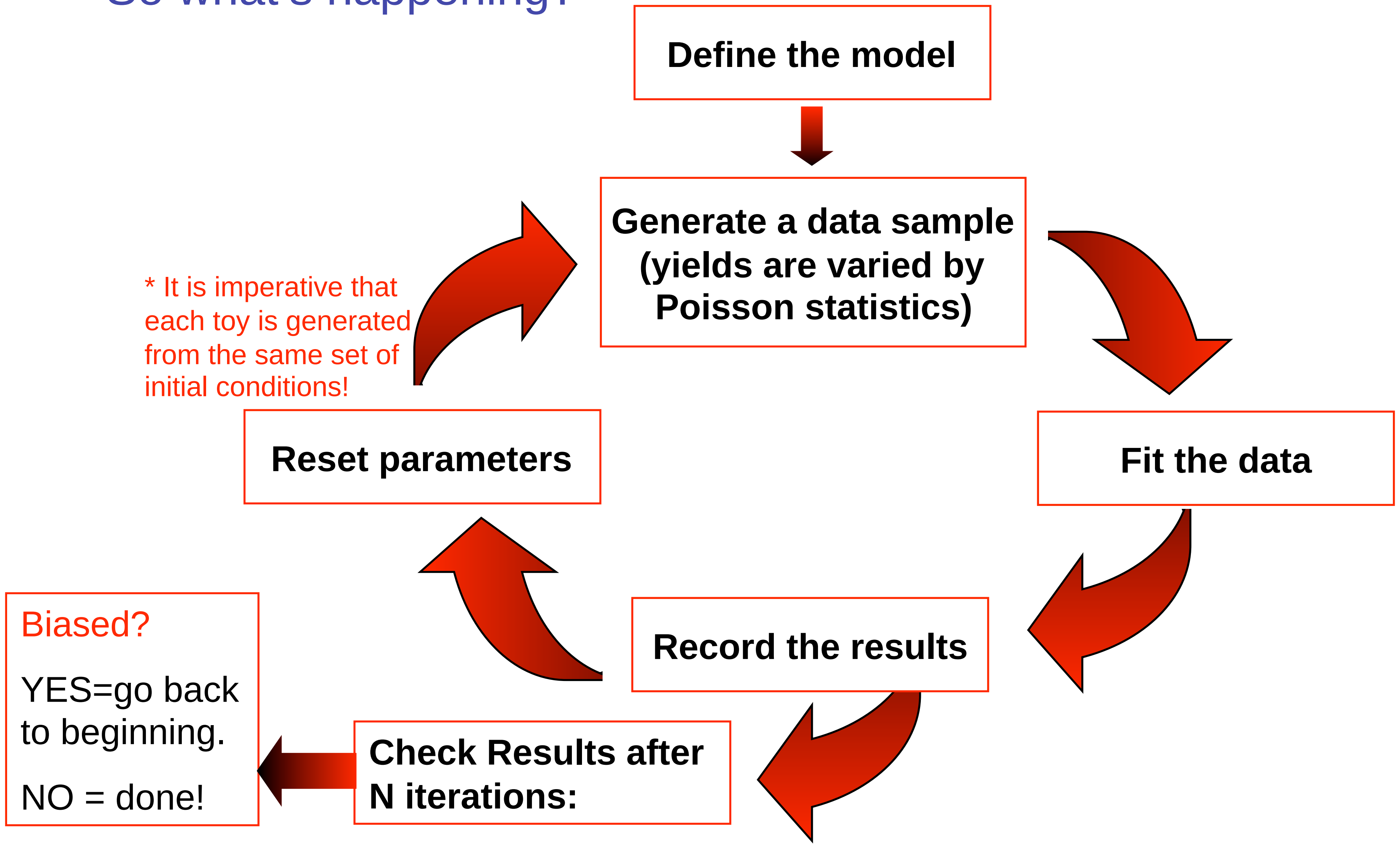
Note for complicated fits, it is more practical to generate and fit samples of events one by one, rather than in a for loop.

It is important to reset parameters to initial values when generating and fitting back in the same loop (like this example).

- A safe way to do this is to remake the PDF from an AFitMaster within the for loop.

Toy MC [PDF Toys]

- So what's happening?


Define the model
Generate a data sample (yields are varied by Poisson statistics)
Fit the data
Record the results
Reset parameters
* It is imperative that each toy is generated from the same set of initial conditions!
Check Results after N iterations:
Biased?
YES=go back to beginning.
NO = done!

Toy MC [Embedded]

- When we neglect correlations between pairs of discriminating variables, we assume that this is a sensible thing to do.
  - We can check this using toy Monte Carlo techniques, by replacing the components generated from the likelihood model with GEANT Monte Carlo simulated events from your experiment.
  - A suitably detailed GEANT Monte Carlo simulation will reconstruct correlations expected in the data for a given component.
  - Want to use control samples of GEANT Monte Carlo, often with events simulated from sub-components of the likelihood model to run ensembles of toy experiments.
  - We call these 'Embedded Toy Monte Carlo Experiments' to distinguish them from PDF toys.

Toy MC [Embedded]

- Several ways to approach the problem using member functions of AFitToy:

- 1) Generate a single sample of events from an existing data set:

```
AFitToy toy;
toy.setPoisson(kFALSE);
```

N.B. For embedded toys we want to test bias from our assumptions, and don't care about making pull plots, so we don't have to generate events with a Poisson variation.

```
RooDataSet * theone = toy.generatePrototype(data, 1000);
```

- 2) Generate a set of samples from an existing data set:

```
toy.generateEmbeddedToys(data, 1000, 0, 10);
```

#events, first toy, last toy

These 11 toy files will be written to a directory (default = ./) which can be specified by the user.

Toy MC [File Management]

- Simple embedded Toy MC will have a signal sample (GEANT MC), and a background sample (PDF toy). These need to be combined before they can be fitted.

```
TString filestomerge = "comma,separated,list";
TObjArray * fArr = filestomerge.Tokenize(',');
toy.mergeFiles(fArr, "myMergedFile.root");
```

- Once the files are merged together, you need to write a macro or programme to fit them. The results can be stored in a TTree using:

```
void AFitToy::dumpFitResult(TString file, RooFitResult * result)
```

where the results file 'file' will be written from the floating parameters (value and parabolic error) extracted from th RooFitResult, global correlation coefficients and the fit status.

- The TTrees can be merged for further processing using:

```
void AFitToy::chainResults(TChain & chain, Int_t imin, Int_t imax,
                                          Bool_t kCutOnStatus)
```

Plotting 1: Projecting over all data

- Project PDF by integrating over all other variables to produce the marginal distribution for x_i:

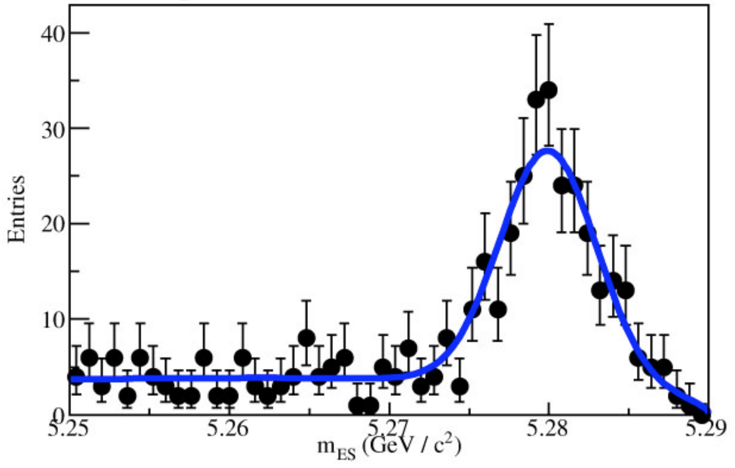
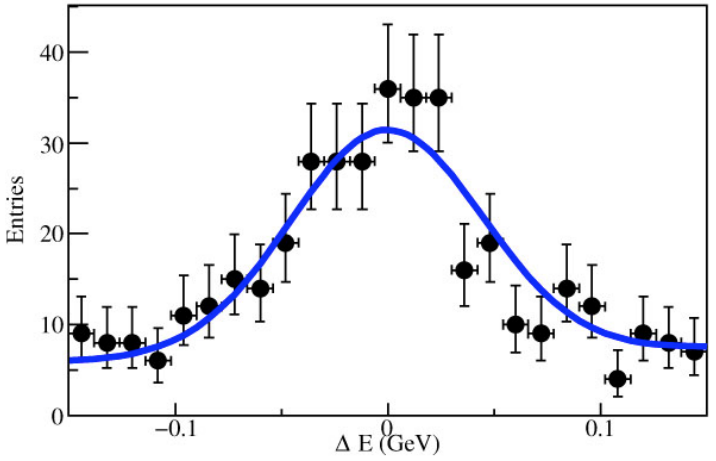
$$\mathcal{P}(x_i) = \int_{all\ space} \mathcal{P}(\underline{x})d\underline{x}'$$

•$\underline{x}$' contains all variables in $\underline{x}$, except x_i.

•The integral is over all space relevant for the fit.

- For example consider an $m_{ES}-\Delta E$ fit:

$$\mathcal{P}(m_{ES}) = \int_{all\ space} \mathcal{P}(m_{ES}, \Delta E)d\Delta E \qquad \mathcal{P}(\Delta E) = \int_{all\ space} \mathcal{P}(m_{ES}, \Delta E)dm_{ES}$$

$\mathcal{P}(\underline{x})$ =  × 

Plotting 1: Projecting over all data

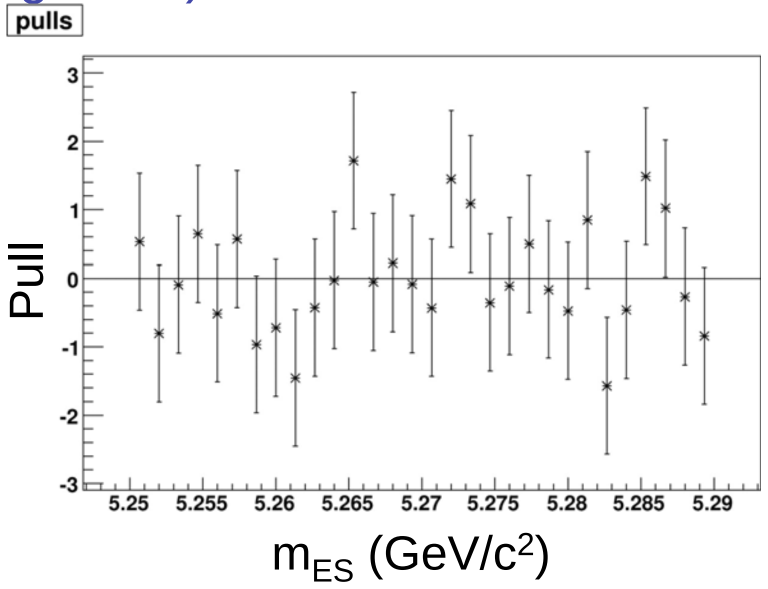
- The `AFitProjectionPlot` class can be used as an interface to RooFit's plotOn function.
- Simple to use: need a discriminating variable, PDF, dataset:

```
AFitProjectionPlot plotter;
RooPlot * frame_bMes = plotter.makePlot(*bMes, data, pdf);
```

- If `data` is null, only plots `pdf`, similarly only plots `data` if `pdf` is null.
- Can make plots using prototype data samples.
- Can also plot a second PDF (e.g. background) on the frame at the same time.
- Also able to make a frame of the pulls of a PDF: i.e. (PDF-data)/σ

```
RooHist * makePullPlot(RooHist*
             dhist, RooCurve* curve);
```

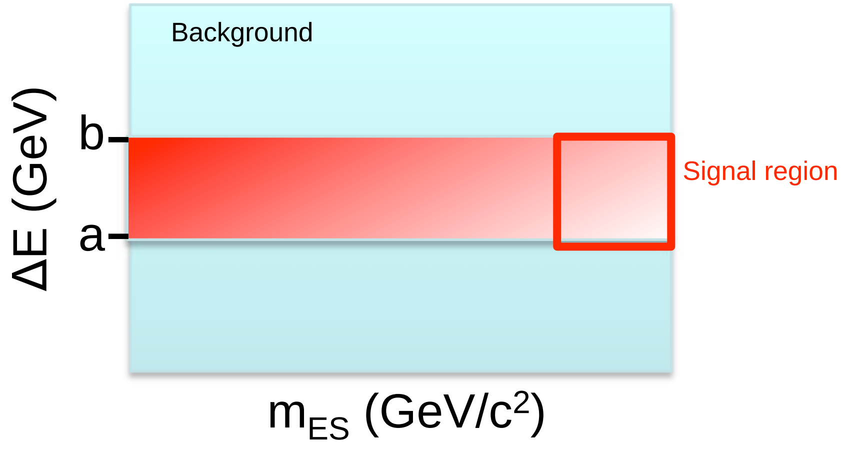
given a RooHist from an existing frame.



Plotting 2: Projecting a slice

- For many fits all space includes background sideband regions that are not interesting from the perspective of understanding signal.
  - Would like to make projections of the 'interesting space' instead of all space.

$$\mathcal{P}(m_{ES}) = \int_a^b \mathcal{P}(m_{ES}, \Delta E)d\Delta E$$



  - To do this you just need to add a string describing the cut you want to make on the data:

```
RooPlot * framecut_bMes = plotter.makePlot("abs(bDeltaE)<0.1", *bMes, data, pdf);
```

Plotting 3: likelihood ratio cut

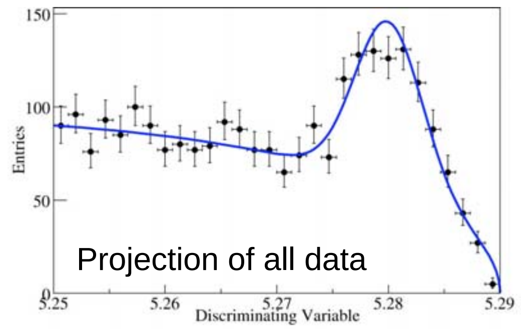
- You can also use an interface to RooFit's plotting functions to simplify the process of making a plot while cutting on a likelihood ratio.
  - Step 1: Set up your PDF to correspond to the result obtained by fitting the data (either from an appropriate data card or by fitting the data itself).
  - Step 2: Compute a likelihood ratio R:

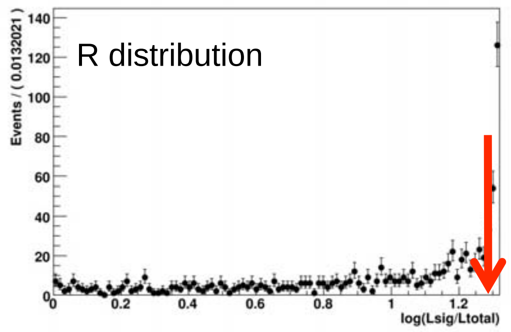
$$R(\underline{x}') = \frac{\mathcal{L}_{signal}(\underline{x}')}{\mathcal{L}_{total}(\underline{x}')}$$

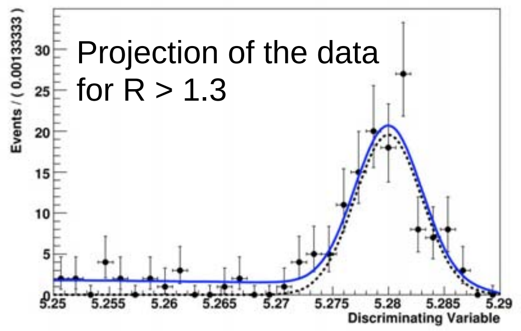
This ratio is computed excluding the variable you wish to project.

  - and make a plot of the distribution of R:
  - Step 3: Determine where to cut on R, and make the projection plot you're interested in.

Plotting 3: likelihood ratio cut







- The original projection shows that there is a signal.
  - But there is a lot of side-band background included in this projection.

- Step 2: Make a plot of R:

```
AFitProjectionPlot plotter;
RooPlot * llrframe = plotter.makeLRProjectionFrame(
                           pdf,data,varToProject,sigCompName);
llrframe->Draw();
```

- Step 2: Make the projection cutting on R:

```
RooPlot * projframe = plotter.makeLRProjection(pdf, data,
                             varToProject, sigCompName, 1.3);
projframe->Draw();
```

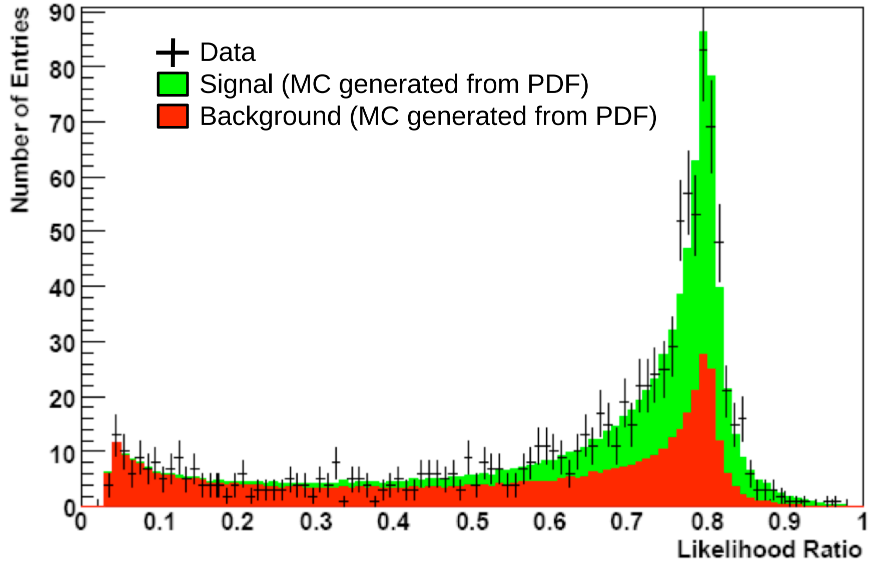
When making a projection ask how you plan to cut on R. Do you find an optimal place to cut, or choose one or more values to demonstrate that the PDF describes the total signal and background, while enhancing the relative signal content of the projection?

Plotting 4: Likelihood Ratio Plots

- A limitation of making a projection while cutting on a likelihood ratio is you don't see how good the agreement is between all events in the data, and your PDF model.
  - Compute the likelihood ratio R integrating over all space, and plot this instead.

$$R(\underline{x}) = \frac{\mathcal{L}_{signal}(\underline{x})}{\mathcal{L}_{total}(\underline{x})}$$

- argSet is a RooArgSet of the variables $\underline{x}$.
- nSig and nBG are signal and background yields.



```
AFitLRPlot lrplot;
lrplot.makePlot(signalPDF, bgPDF, argSet, data, nSig, nBG);
```

- Any significant discrepancy between data and MC would indicate a problem with the model assumed.

Summary

- *A*Fit has a number of utilities to facilitate defining a likelihood fit and validating it.
  - Statistical tools for testing correlations between input variables (and other calculations).
  - Interface to TMVA in order to define your favourite multivariate analyser, and append this variable toy your data and Monte Carlo Files.
  - An interface class to facilitate running pure and embedded toy Monte Carlo experiments.
  - Plotting interfaces to RooFit.