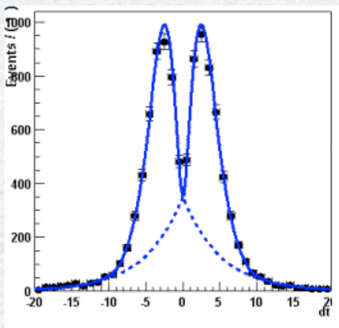
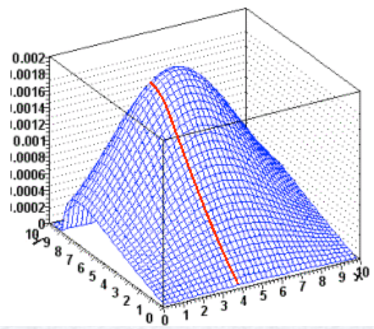
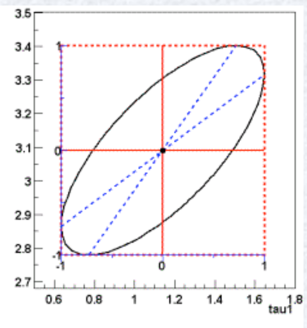
Statistical Software Tools
RooFit/RooStats

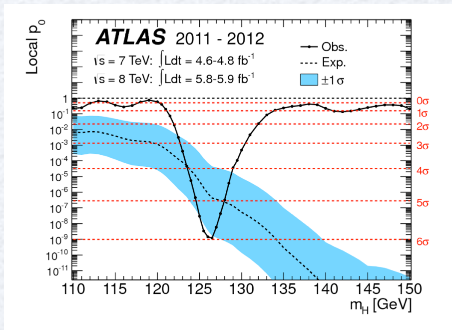
Lorenzo Moneta (CERN)

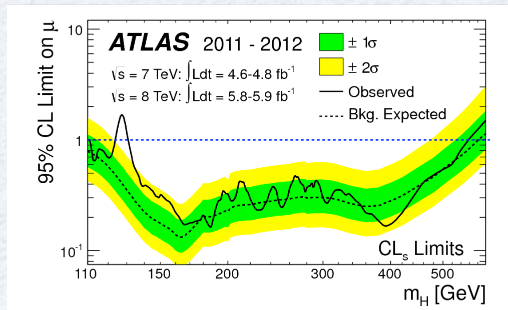
Terascale Statistics School 2015











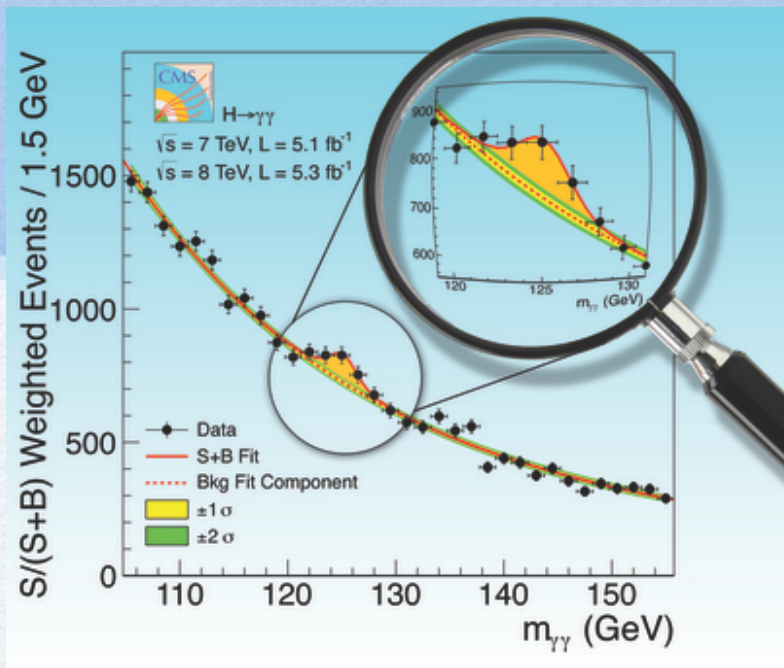
Introduction

- We will cover only RooFit/RooStats
- Statistical tools for:
  - point estimation: determine the best estimate of a parameter
  - estimation of confidence (credible) intervals
    - lower/upper limits or multi-dimensional contours
  - hypothesis tests:
    - evaluation of p-value for one or multiple hypotheses (discovery significance)
- Model description and sharing of results
  - analysis combination

Outline

- Today:
  - Introduction to Fitting in ROOT
  - Model building and parameter estimation in RooFit
  - Exercises
- Tomorrow
  - Introduction to RooStats
  - Interval estimation tools (Likelihood/Bayesian)
  - Hypothesis tests
  - Frequentist interval/limit calculation (CLs)
  - Exercises
- Thursday
  - Tutorial on building model with the HistFactory

Introduction to Fitting in ROOT

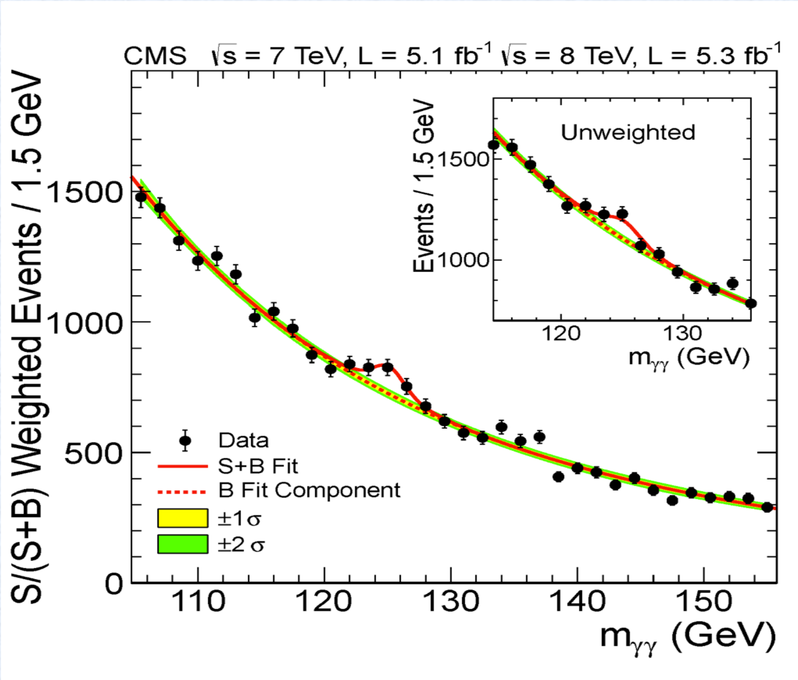


Outline

- Introduction to Fitting:
  - fitting methods in ROOT
  - how to fit a histogram in ROOT,
  - how to retrieve the fit result.
- Building fit functions in ROOT.
- Interface to Minimization.
- Common Fitting problems.
- Using the ROOT Fit GUI (Fit Panel).

What is Fitting ?

- Estimate parameters of an hypothetical distribution from the observed data distribution
  - $y = f(x \mid \theta)$ is the fit model function
- Find the best estimate of the parameters θ assuming $f(x \mid \theta)$



Example

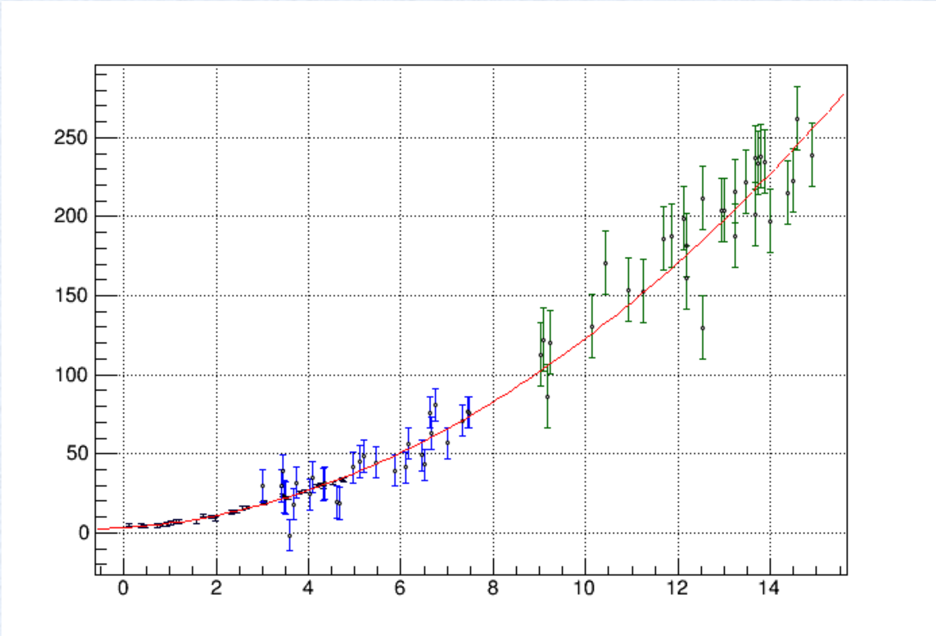
Higgs → $\gamma\gamma$ spectrum

We can fit for:

- the expected number of Higgs events
- the Higgs mass

Least Square (χ^2) Fit

- Minimizes the deviations between the observed y and the predicted function values:
- Least square fit (χ^2) : minimize square deviation weighted by the errors
  - observed errors (Neyman χ^2)
    - $\sigma_i = \sqrt{N_i}$ for the histograms
  - expected errors (Pearson χ^2)
    - $\sigma_i = \sqrt{f(X_i, \theta)}$



$$\chi^2 = \sum_i \frac{(Y_i - f(X_i, \theta))^2}{\sigma_i^2}$$

Maximum Likelihood Fit

- The parameters are estimated by finding the maximum of the likelihood function (or minimum of the negative log-likelihood function).
  - Likelihood:

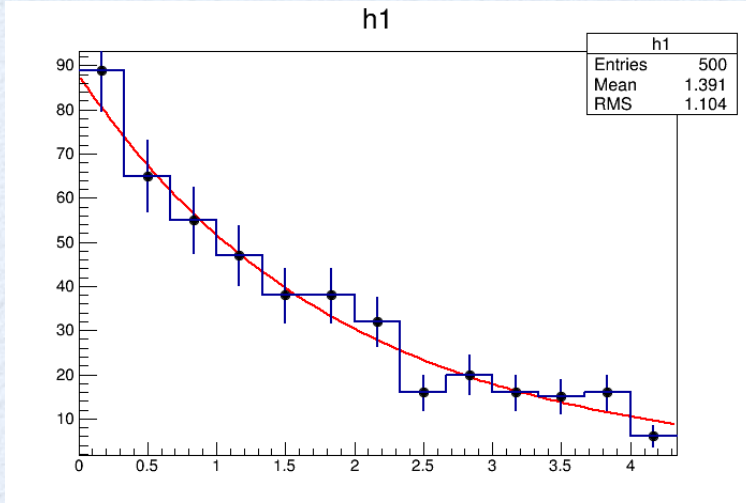
$$L(x|\theta) = \prod_i P(x_i|\theta)$$

  - Find best value θ, the maximum of

$$logL = \sum_i \log(f(x_i, \theta)$$

- The least-square fit and the maximum likelihood fit are equivalent when the distribution of observed events in each bin is normal.
  - $f(x \mid \theta)$ is gaussian

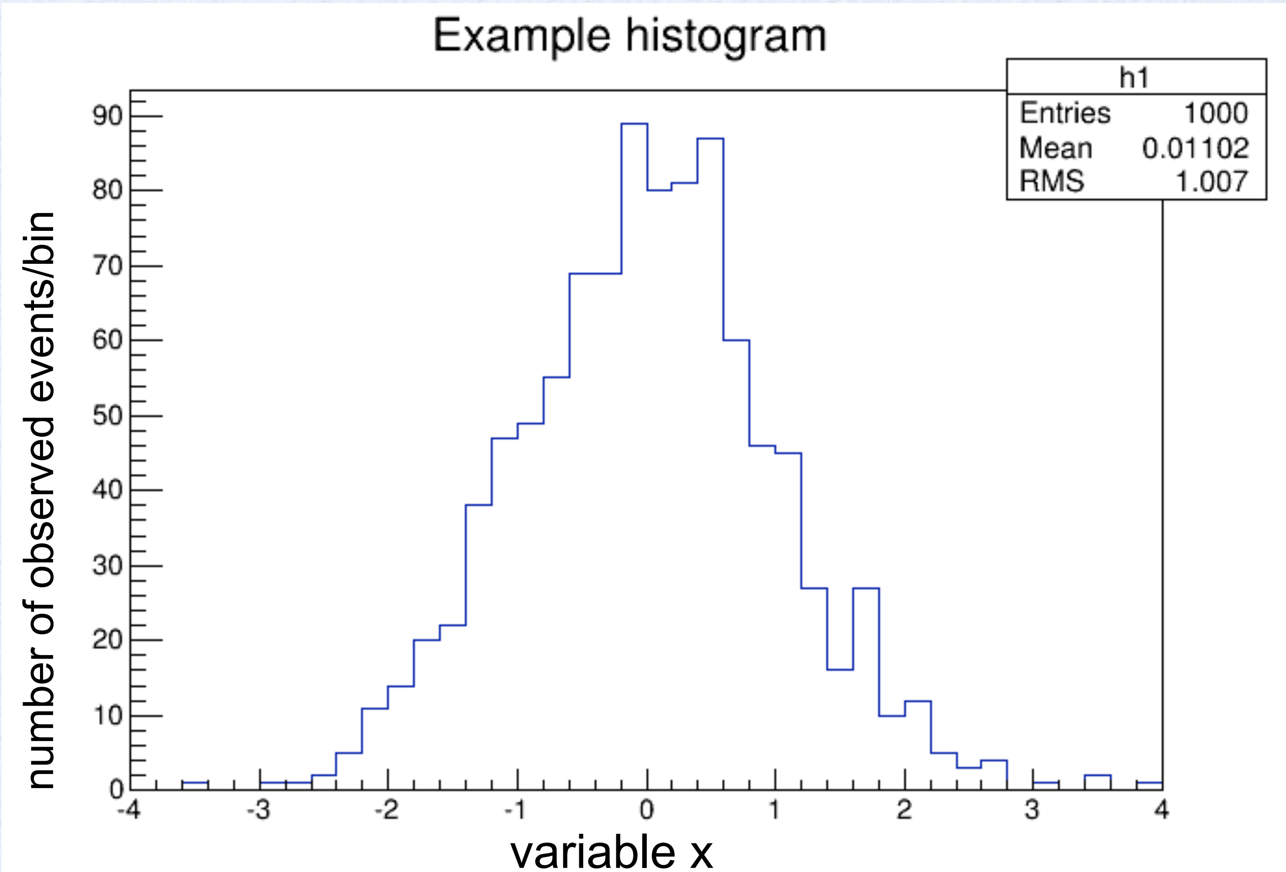
ML Fit of an Histogram



- The Likelihood for a histogram is obtained by assuming a Poisson distribution in every bin:
  - **Poisson($n_i|\nu_i$)**
    - n_i is the observed bin content.
    - ν_i is the expected bin content,
      $\nu_i = f\ (x_i|\theta)$, where x_i is the bin center, assuming a linear function within the bin. Otherwise it is obtained from the integral of the function in the bin.
- For large histogram statistics (large bin contents) bin distribution can be considered normal
  - equivalent to least square fit
- For low histogram statistics the ML method is the correct one !

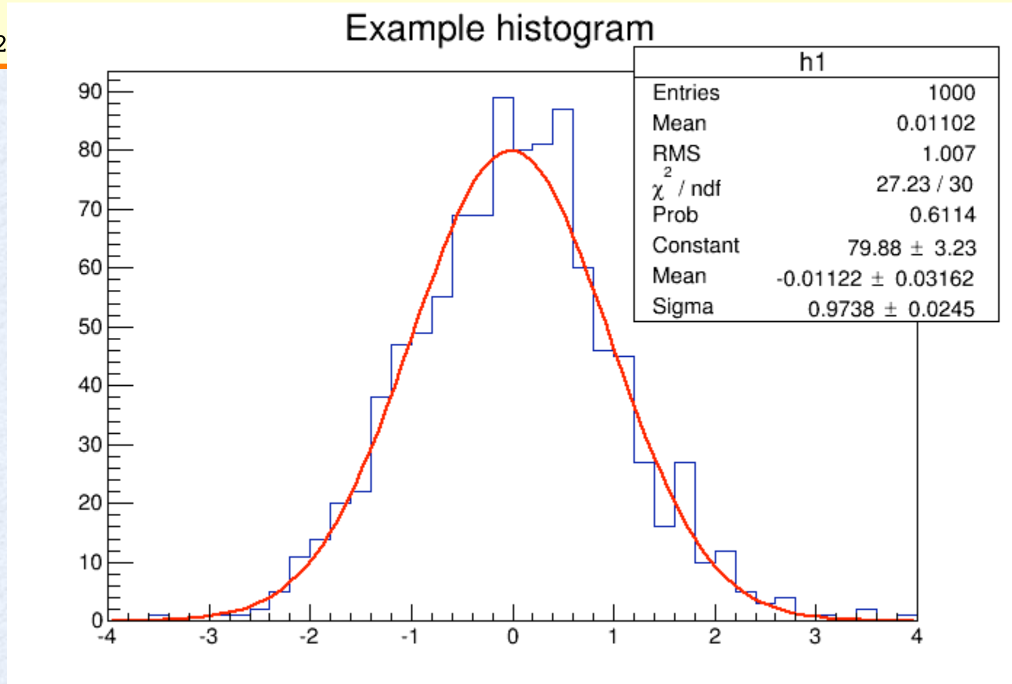
Simple Gaussian Fitting

- Suppose we have this histogram
  - we want to estimate the mean and sigma of the underlying gaussian distribution.



Fitting Histogram

```
root [] TF1 * f1 = new TF1("f1","gaus");
root [] f1->SetParameters(1,0,1);
root [] h1->Fit(f1);
 FCN=27.2252 FROM MIGRAD    STATUS=CONVERGED      60 CALLS          61 TOTAL
                     EDM=1.12393e-07    STRATEGY= 1      ERROR MATRIX ACCURATE
  EXT PARAMETER                                  STEP         FIRST
  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
   1  Constant     7.98760e+01   3.22882e+00   6.64363e-03  -1.55477e-05
   2  Mean        -1.12183e-02   3.16223e-02   8.18642e-05  -1.49026e-02
   3  Sigma        9.73840e-01   2.44738e-02   1.692
```



For displaying the fit parameters:

```
gStyle->SetOptFit(1111);
```

Creating the Fit Function

- To create a parametric function object (a **`TF1`**) :
    - we can use the available functions in ROOT library

```
TF1 * f1 = new TF1("f1","[0]*TMath::Gaus(x,[1],[2])");
```

    - and also use it to write formula expressions
        - [0],[1],[2] indicate the parameters
    - we can also use pre-defined functions

```
TF1 * f1 = new TF1("f1","gaus");
```

    - using pre-defined functions we have the parameter name automatically set to meaningful values.
    - initial parameter values are estimated whenever possible.
    - pre-defined functions avalaible:
        - gaus, expo, landau, pol0,1..,10, chebyshev

Building More Complex Functions

- Sometimes better to write directly the functions in C/C++
  - but in this case object cannot be fully stored to disk
- Using a general free function with parameters:

```
double function(double *x, double *p){
   return p[0]*TMath::Gaus(x[0],p[0],p[1]);
}
TF1 * f1 = new TF1("f1",function,xmin,xmax,npar);
```

- any C++ object implementing double operator() (double *x, double *p)

```
struct Function {
   double operator()(double *x, double *p){
      return p[0]*TMath::Gaus(x[0],p[0],p[1]);}
};
Function func;
TF1 * f1 = new TF1("f1",&func,xmin,xmax,npar,"Function");
```

  - e.g using a lambda function (with Cling and C++-11)

```
auto f1 = new TF1("f1",[](double *x, double *p){return p[0]*x[0];},0,10,1);
```

Retrieving The Fit Result

- The main results from the fit are stored in the fit function, which is attached to the histogram; it can be saved in a file (except for C/C++ functions were only points are saved).
- The fit function can be retrieved using its name:

```
TF1 * fitFunc = h1->GetFunction("f1");
```

- The parameter values/error using indices (or their names):

```
fitFunc->GetParameter(par_index);
fitFunc->GetParError(par_index);
```

- It is also possible to access the `TFitResult` class which has all information about the fit, if we use the fit option "S":

```
TFitResultPtr r = h1->Fit(f1,"S");
r->Print();
TMatrixDSym C = r->GetCorrelationMatrix();
```

C++ Note: the TFitResult class is accessed by using operator-> of TFitResultPtr

Some Fitting Options

- Fitting in a Range

```
h1->Fit("gaus","","",-1.5,1.5);
```

- Quite / Verbose: option "Q" /"V".

```
h1->Fit("gaus","V");
```

- Likelihood fit for histograms
  - option "L" for count histograms;

```
h1->Fit("gaus","L");
```

  - option "WL" in case of weighted counts.

```
h1->Fit("gaus","LW");
```

- Default is chi-square with observed errors (and skipping empty bins)
  - option "P" for Pearson chi-square (expected errors) with empty bins

```
h1->Fit("gaus","P");
```

- Use integral function of the function in bin

```
h1->Fit("gaus","L I");
```

- Compute MINOS errors : option "E"

```
h1->Fit("gaus","L E");
```

All fitting options documented in reference guide or User Guide (Fitting Histogram chapter)

Note on Binned Likelihood Fit

- Log-Likelihood is computed using Baker-Cousins procedure (Likelihood χ^2)

$$\chi^2_\lambda(\theta) = -2\ln\lambda(\theta) = 2\sum_i [\mu_i(\theta) - n_i + n_i \ln(n_i/\mu_i(\theta))]$$

  - $-2\ln\lambda(\theta)$ is an equivalent chi-square
  - Its value at the minimum can be used for checking the fit quality
    - avoiding problems with bins with low content
- ROOT computes $-\ln\lambda(\theta)$
  - retrieve it using `TFitResult::MinFcnValue()`

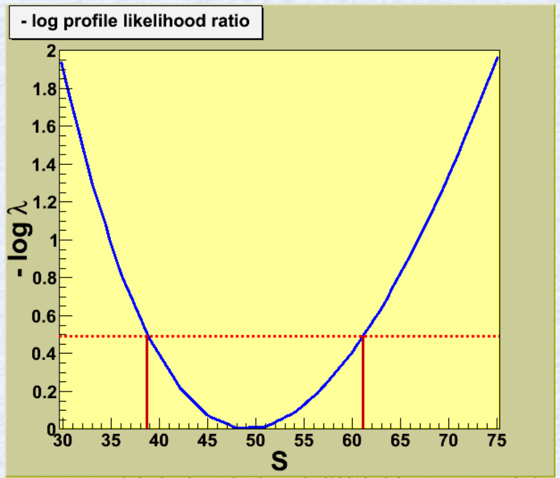
Parameter Errors

- Errors returned by the fit are computed from the second derivatives of the likelihood function
  - Asymptotically the parameter estimates are normally distributed. The estimated correlation matrix is then:

$$\hat{\mathbf{V}}(\hat{\boldsymbol{\theta}}) = \left[\left(- \frac{\partial^2 \ln L(\mathbf{x};\boldsymbol{\theta})}{\partial^2 \boldsymbol{\theta}} \right)_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}} \right]^{-1} = \mathbf{H}^{-1}$$

- A better approximation to estimate the confidence level in the parameter is to use directly the log-likelihood function and look at the difference from the minimum.
  – Method of Minuit/Minos (Fit option "E")
    – obtain a confidence interval which is in general not symmetric around the best parameter estimate

```
TFitResultPtr r = h1->Fit(f1,"E S");
r->LowerError(par_number);
r->UpperError(par_number);
```



Minimization

- The fit is done by minimizing the least-square or likelihood function.
- A direct solution exists only in case of linear fitting
  - it is done automatically in such cases (e.g fitting polynomials).
- Otherwise an iterative algorithm is used:
  - Minuit is the minimization algorithm used by default
    - ROOT provides two implementations: Minuit and Minuit2
    - other algorithms exists: Fumili, or minimizers based on GSL, genetic and simulated annealing algorithms
  - To change the minimizer:

```
ROOT::Math::MinimizerOptions::SetDefaultMinimizer(“Minuit2”);
```

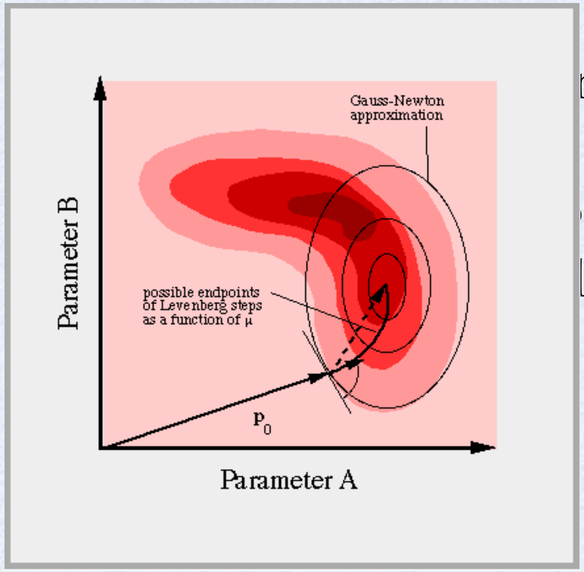
  - Other commands are also available to control the minimization:

```
ROOT::Math::MinimizerOptions::SetDefaultTolerance(1.E-6);
```

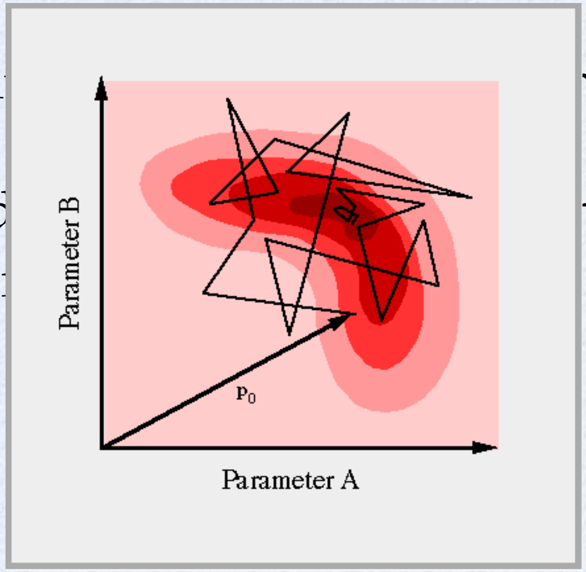
Minimization Techniques

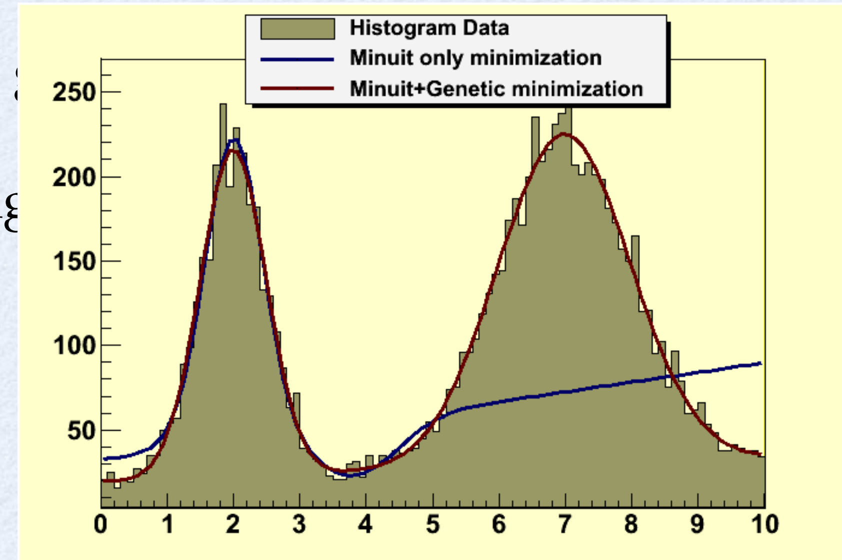
Example: Fitting 2 peaks in a spectrum

Quadratic Newton



Simulated Annealing





Function Minimization

- Common interface class (`ROOT::Math::Minimizer`)
- Existing implementations available as plug-ins:
  - **Minuit** (based on class `TMinuit`, direct translation from Fortran code)
    - with Migrad, Simplex, Minimize algorithms
  - **Minuit2** (new C++ implementation with OO design)
    - with Migrad, Simplex, Minimize and Fumili2
  - **Fumili** (only for least-square or log-likelihood minimizations)
  - **GSLMultiMin**: conjugate gradient minimization algorithm from GSL (Fletcher-Reeves, BFGS)
  - **GSLMultiFit**: Levenberg-Marquardt (for minimizing least square functions) from GSL
  - **Linear** for least square functions (direct solution, non-iterative method)
  - **GSLSimAn**: Simulated Annealing from GSL
  - **Genetic**: based on a genetic algorithm implemented in TMVA
- All these are available for ROOT fitting and in RooFit/RooStats
- Possible to combine them (e.g. use Minuit and Genetic)
- Easy to extend and add new implementations
  - e.g. minimizer based on NagC exists in the development branch (see here)

Comments on Minimization

- Sometimes fit converges to a wrong solution
  - Often is the case of a local minimum which is not the global one.
    - This is often solved with better initial parameter values. A minimizer like Minuit is able to find only the local best minimum using the function gradient.
    - Otherwise one needs to use a genetic or simulated annealing minimizer (but it can be quite inefficient, e.g. many function calls).
- Sometimes fit does not converge

```
Warning in <Fit>: Abnormal termination of minimization.
```

  - can happen because the Hessian matrix is not positive defined
    - e.g. there are no minimum in that region →wrong initial parameters;
  - numerical precision problems in the function evaluation
    - need to check and re-think on how to implement better the fit model function;
  - highly correlated parameters in the fit. In case of 100% correlation the point solution becomes a line (or an hyper-surface) in parameter space. The minimization problem is no longer well defined.

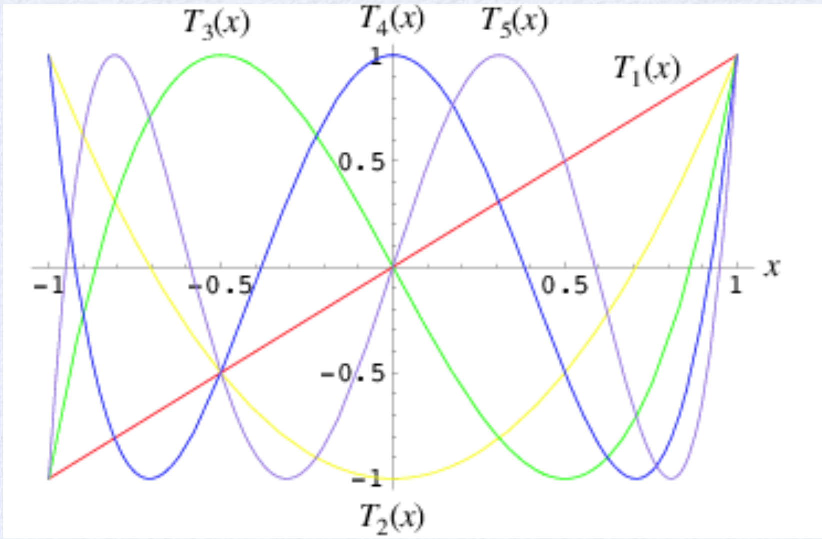
```
PARAMETER  CORRELATION COEFFICIENTS
      NO.  GLOBAL      1      2
        1  0.99835   1.000  0.998
        2  0.99835   0.998  1.000
```

Signs of trouble...

Mitigating fit stability problems

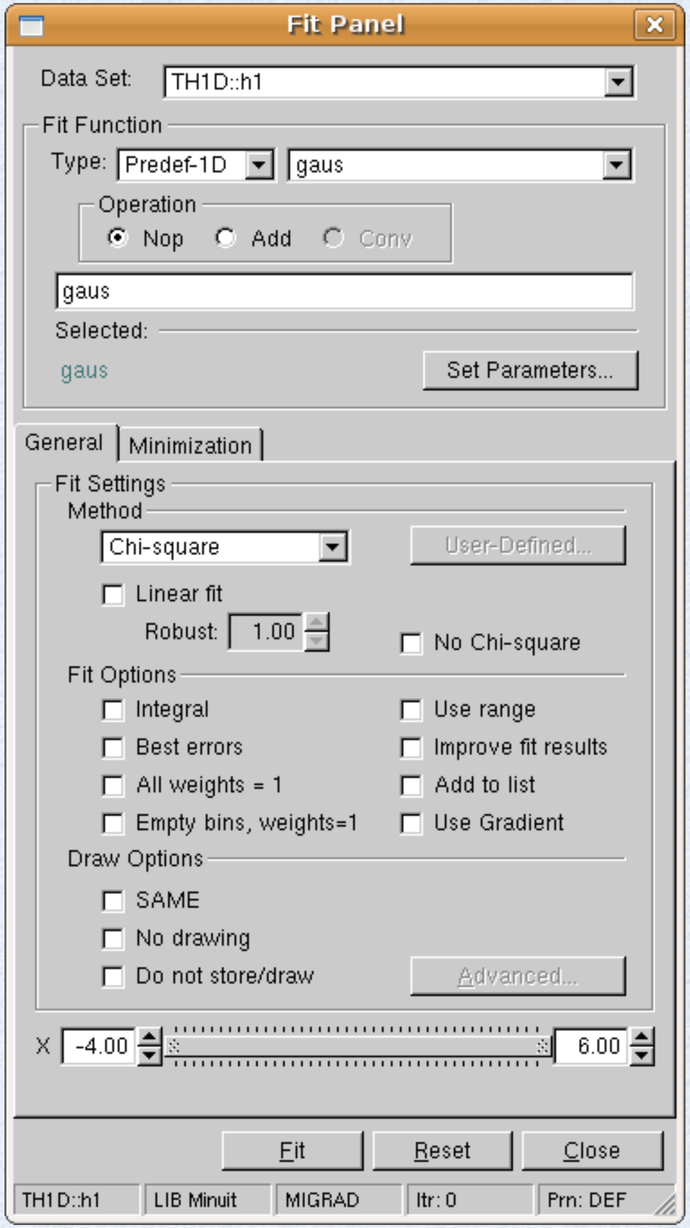
- When using a polynomial parametrization:
  - $a_0+a_1x+a_2x^2+a_3x^3$ nearly always results in strong correlations between the coefficients.
    - problems in fit stability and inability to find the right solution at high order
- This can be solved using a better polynomial parametrization:
  - e.g. Chebychev polynomials

$$
\begin{aligned}
T_0(x) &= 1 \\
T_1(x) &= x \\
T_2(x) &= 2x^2 - 1 \\
T_3(x) &= 4x^3 - 3x \\
T_4(x) &= 8x^4 - 8x^2 + 1 \\
T_5(x) &= 16x^5 - 20x^3 + 5x \\
T_6(x) &= 32x^6 - 48x^4 + 18x^2 - 1.
\end{aligned}
$$



The Fit Panel

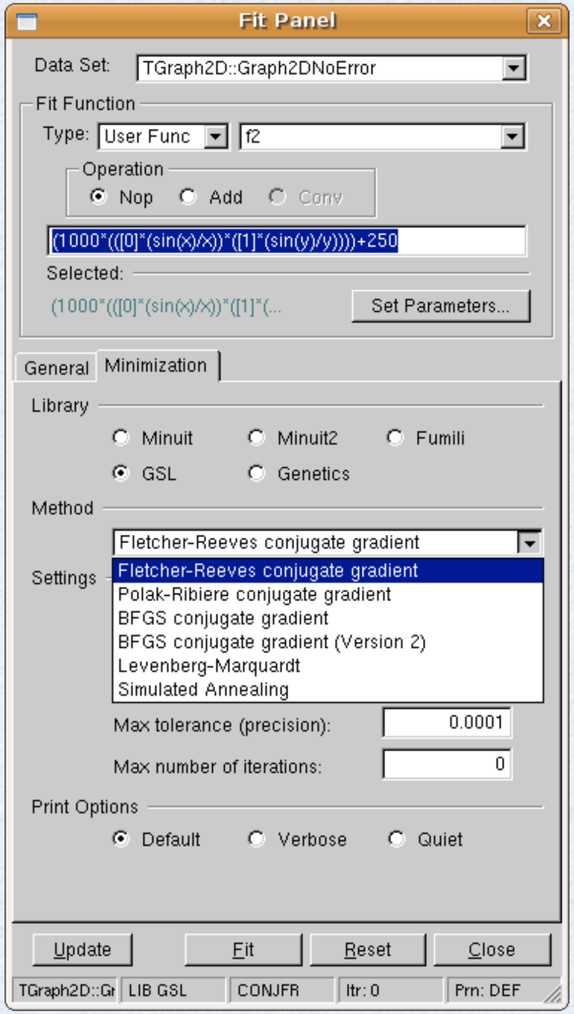
- The fitting in ROOT using the FitPanel GUI
  - GUI for fitting all ROOT data objects (histogram, graphs, trees)
- Using the GUI we can:
  - select data object to fit
  - choose (or create) fit model function
  - set initial parameters
  - choose:
    - fit method (likelihood, chi2)
    - fit options (e.g Minos errors)
    - drawing options
  - change the fit range



Fit Panel (2)

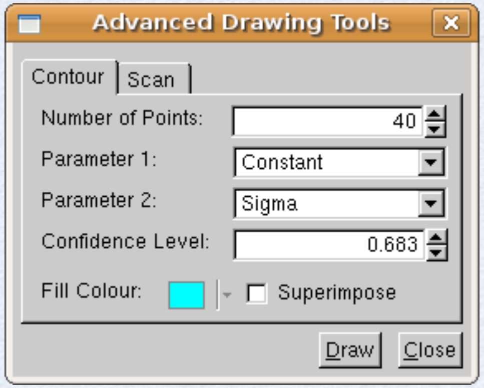
- The Fit Panel provides also extra functionality:

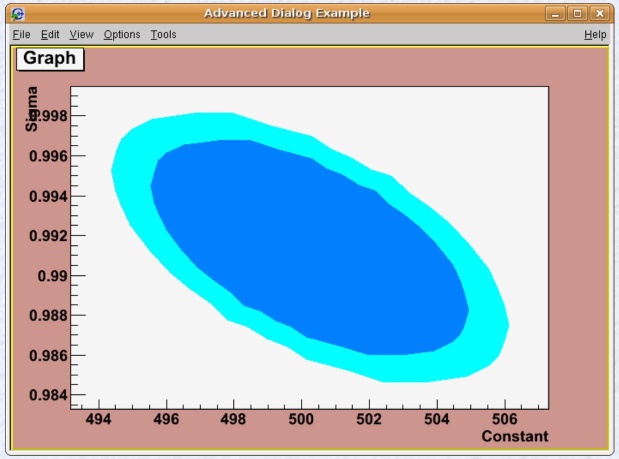
Control the minimization



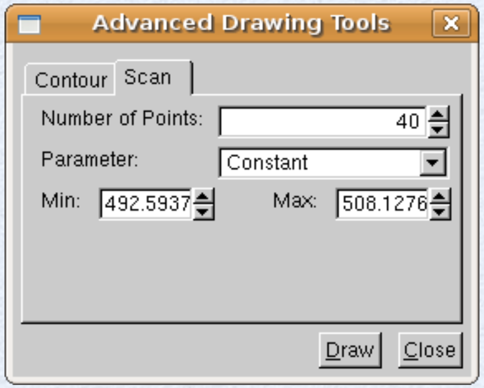
Advanced drawing tools

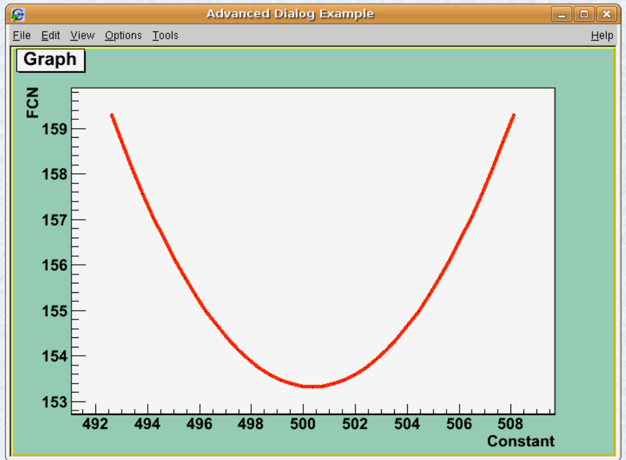
Contour plot



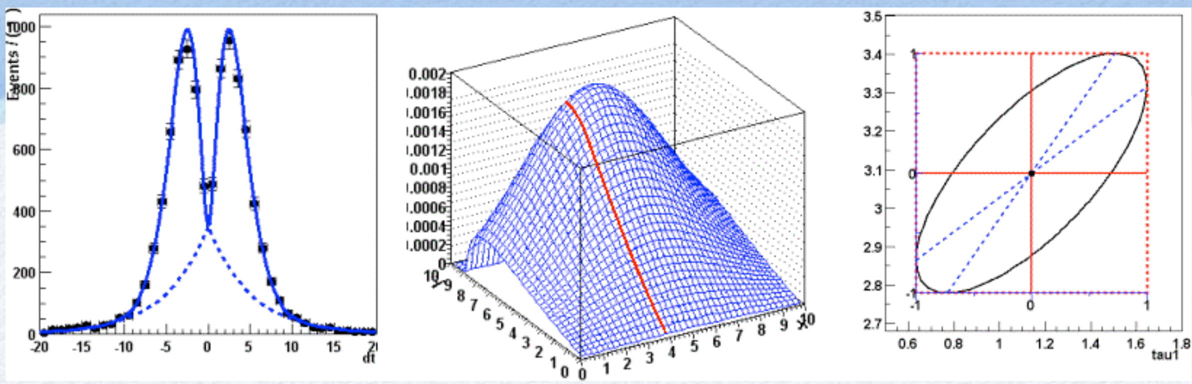


Scan plot of minimization function





RooFit



Events / (1)
1000
800
600
400
200
0
-20
-15
-10
-5
0
5
10
15
20
dt
0.002
0.0018
0.0016
0.0014
0.0012
0.001
0.0008
0.0006
0.0004
0.0002
0
Y
X
3.5
3.4
3.3
3.2
3.1
3
2.9
2.8
2.7
0.6
0.8
1
1.2
1.4
1.6
1.8
tau1

Outline

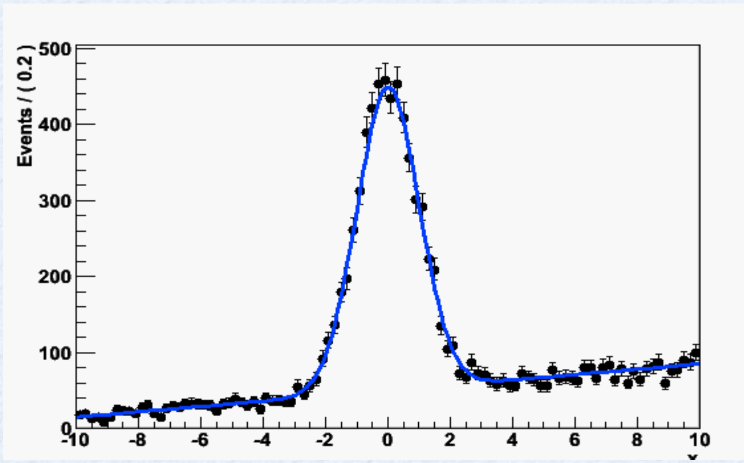
- Introduction to RooFit
  - Basic functionality
  - Model building using the workspace
  - Composite models

Material based on slides from W. Verkerke (author of RooFit)

- Exercises on RooFit:
  - building and fitting model

RooFit

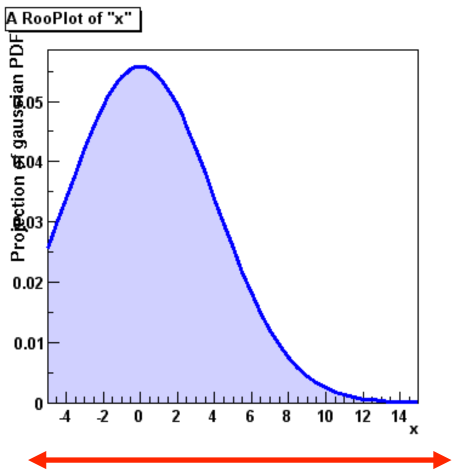
- Toolkit for data modeling
  - developed by *W. Verkerke and D. Kirkby*
- model distribution of observable ***x*** in terms of parameters ***p***
  - probability density function (pdf): $\mathcal{P}$`(x;p)`
- pdf are normalized over allowed range of observables ***x*** with respect to the parameters ***p***



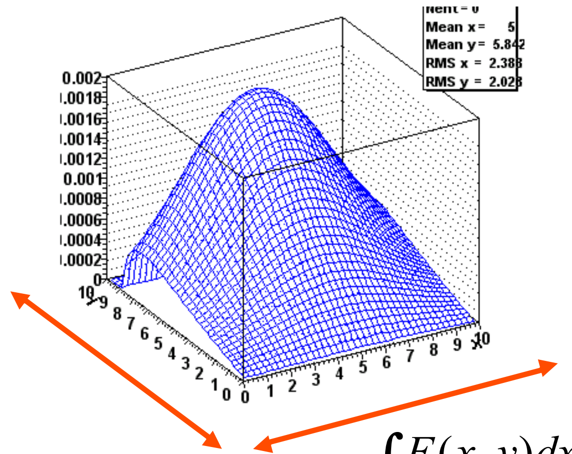
Mathematic – Probability density functions

- Probability Density Functions describe probabilities, thus
  - All values most be >0
  - The total probability must be 1 *for each* *p*, i.e.
  - Can have any number of dimensions

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



$$\int F(x)dx \equiv 1$$



$$\int F(x,y)dxdy \equiv 1$$

- Note distinction in role between *parameters* (p) and *observables* (x)
  - Observables are measured quantities
  - Parameters are degrees of freedom in your model

Why RooFit ?

- ROOT function framework can handle complicated functions
  - but require writing large amount of code
- Normalization of p.d.f. not always trivial
  - RooFit does it automatically
- In complex fit, computation performance important
  - need to optimize code for acceptable performance
  - built-in optimization available in RooFit
    - evaluation only when needed
- Simultaneous fit to different data samples
- Provide full description of model for further use

RooFit

- RooFit provides functionality for building the pdf's
  - complex model building from standard components
  - composition with addition product and convolution
- All models provide the functionality for
  - maximum likelihood fitting
  - toy MC generator
  - visualization

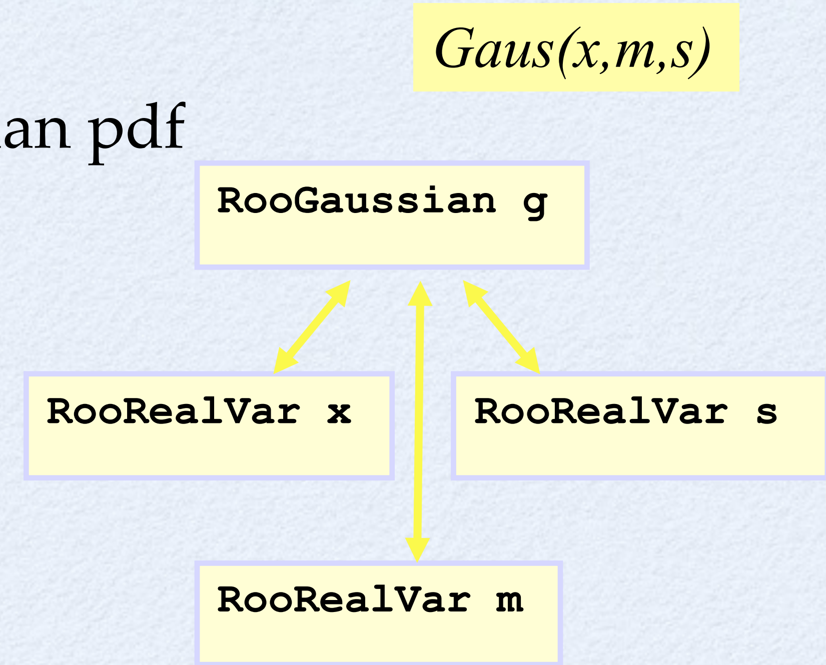
RooFit Modeling

Mathematical concepts are represented as C++ objects

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

RooFit Modeling

Example: Gaussian pdf



RooFit code:

```
RooRealVar x(“x”,”x”,2,-10,10)
RooRealVar s(“s”,”s”,3) ;
RooRealVar m(“m”,”m”,0) ;
RooGaussian g(“g”,”g”,x,m,s)
```

- Represent relations between variables and functions as client/server links between objects

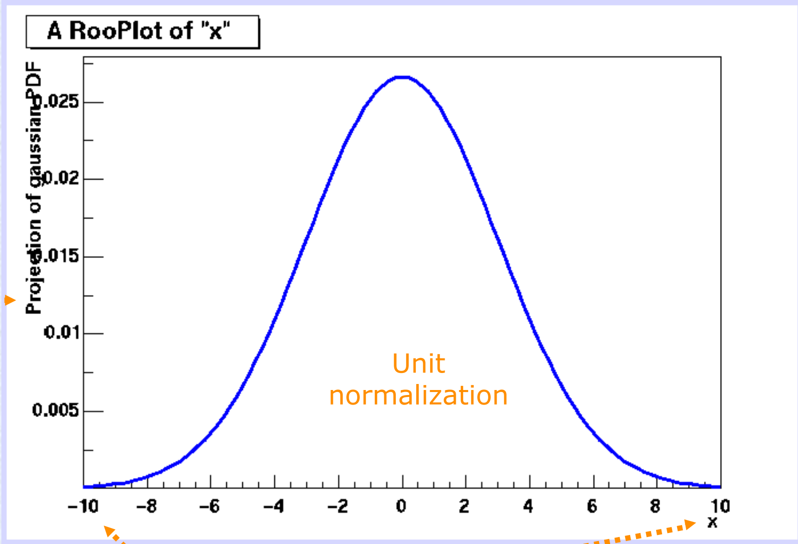
RooFit Functionality

- pdf visualization

```
RooPlot * xframe = x->frame();
pdf->plotOn(xframe);
xframe->Draw();
```

Axis label from **gauss** title

A **RooPlot** is an empty frame capable of holding anything plotted versus it variable



Plot range taken from limits of **x**

RooFit Functionality

- Toy MC generation from any pdf

Generate 10000 events from Gaussian p.d.f and show distribution

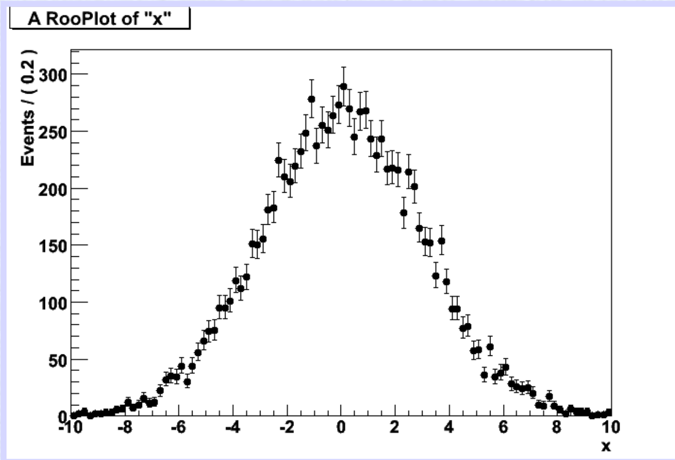
```
RooDataSet * data = pdf->generate(*x,10000);
```

- data visualization

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

Note that dataset is ***unbinned*** (vector of data points, x, values)

Binning into histogram is performed in `data->plotOn()` call



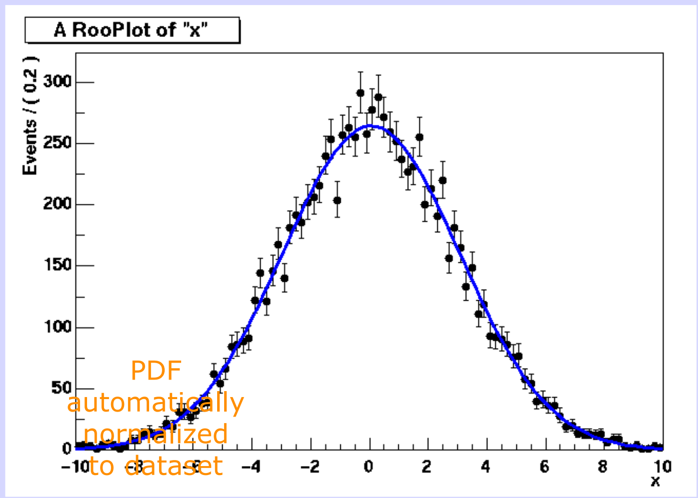
RooFit Functionality

- Fit of model to data
  - e.g. unbinned maximum likelihood fit

```
pdf = pdf->fitTo(data);
```

- data and pdf visualization after fit

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



RooFit Workspace

- `RooWorkspace` class: container for all objected created:
  - full model configuration
    - PDF and parameter/observables descriptions
    - uncertainty/shape of nuisance parameters
  - (multiple) data sets
- Maintain a complete description of all the model
  - possibility to save entire model in a ROOT file
  - all information is available for further analysis
- Combination of results joining workspaces in a single one
  - common format for combining and sharing physics results

```
RooWorkspace workspace(“w”);
workspace.import(*data);
workspace.import(*pdf);
workspace.writeToFile(“myWorkspace.root”)
```

RooFit Factory

```
RooRealVar x("x","x",2,-10,10)
RooRealVar s("s","s",3) ;
RooRealVar m("m","m",0) ;
RooGaussian g("g","g",x,m,s)
```

The workspace provides a factory method to auto-generates objects from a math-like language
(the p.d.f is made with 1 line of code instead of 4)

```
RooWorkspace w;
w.factory("Gaussian::g(x[2,-10,10],m[0],s[3])")
```

In the tutorial we will work using the workspace factory to build models

Using the workspace

- Workspace
  - A generic container class for all RooFit objects of your project
  - Helps to organize analysis projects
- Creating a workspace

```
RooWorkspace w("w") ;
```

- Putting variables and function into a workspace
  - When importing a function or pdf, all its components (variables) are automatically imported too

```
RooRealVar x("x","x",-10,10) ;
RooRealVar mean("mean","mean",5) ;
RooRealVar sigma("sigma","sigma",3)  ;
RooGaussian f("f","f",x,mean,sigma) ;

// imports f,x,mean and sigma
w.import(f) ;
```

Using the workspace

- Looking into a workspace

```
w.Print() ;

variables
---------
(mean,sigma,x)

p.d.f.s
-------
RooGaussian::f[ x=x mean=mean sigma=sigma ] = 0.249352
```

- Getting variables and functions out of a workspace

```
// Variety of accessors available
RooRealVar * x = w.var("x");
RooAbsPdf * f = w.pdf("f");
```

- Writing workspace and contents to file

```
w.writeToFile("wspace.root") ;
```

Factory syntax

- Rule #1 – Create a variable

```
x[-10,10]    // Create variable with given range
x[5,-10,10]  // Create variable with initial value and range
x[5]         // Create initially constant variable
```

- Rule #2 – Create a function or pdf object

```
ClassName::Objectname(arg1,[arg2],...)
```

  - Leading 'Roo' in class name can be omitted
  - Arguments are names of objects that already exist in the workspace
  - Named objects must be of correct type, if not factory issues error
  - Set and List arguments can be constructed with brackets {}

```
Gaussian::g(x,mean,sigma)
         → RooGaussian("g","g",x,mean,sigma)

Polynomial::p(x,{a0,a1})
         → RooPolynomial("p","p",x",RooArgList(a0,a1));
```

Factory syntax

- Rule #3 – Each creation expression returns the name of the object created
  - Allows to create input arguments to functions 'in place' rather than in advance

```
Gaussian::g(x[-10,10],mean[-10,10],sigma[3])
    →  x[-10,10]
       mean[-10,10]
       sigma[3]
       Gaussian::g(x,mean,sigma)
```

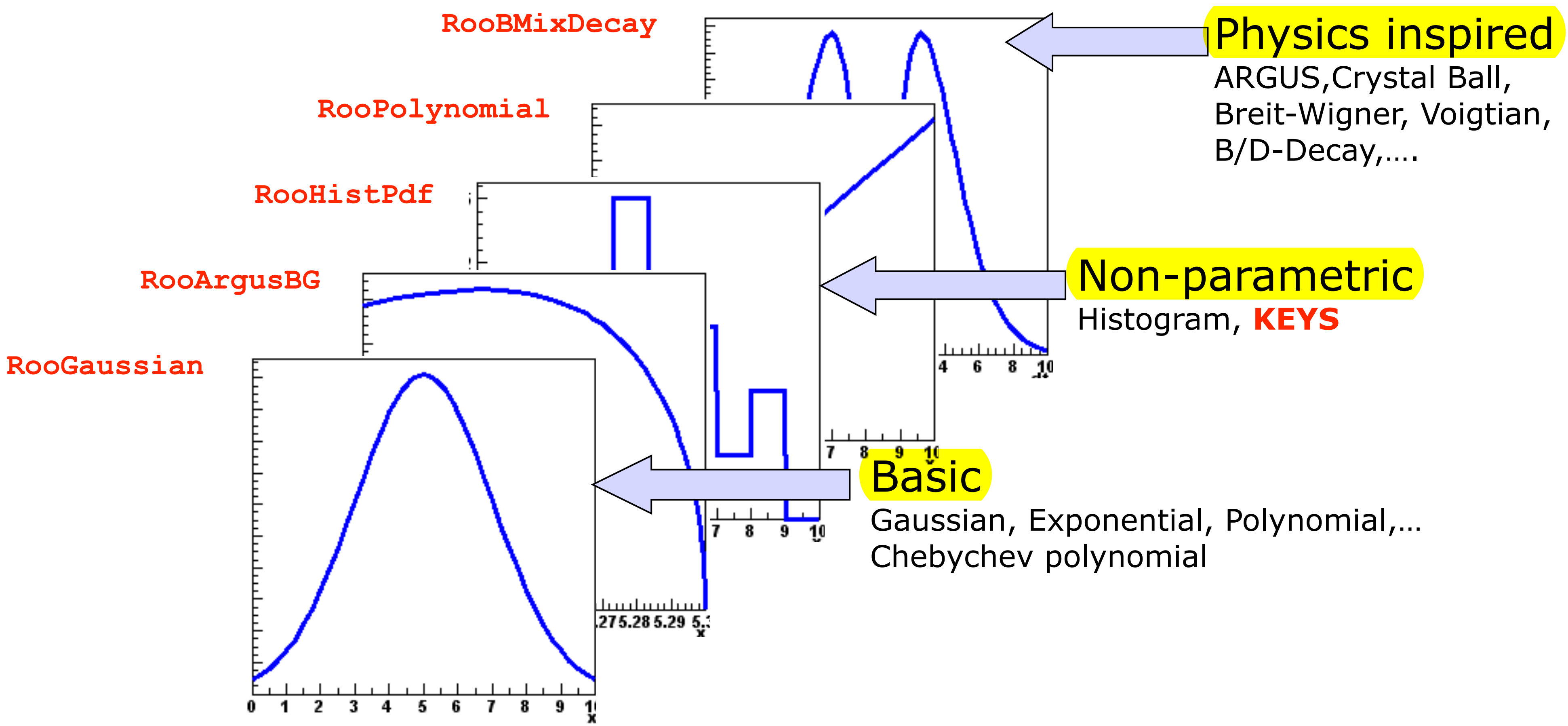
- Miscellaneous points
  - You can always use numeric literals where values or functions are expected
  - It is not required to give component objects a name, e.g.

```
Gaussian::g(x[-10,10],0,3)
```

```
SUM::model(0.5*Gaussian(x[-10,10],0,3),Uniform(x)) ;
```

Model building – (Re)using standard components

- RooFit provides a collection of compiled standard PDF classes



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

Model building – (Re)using standard components

- List of most frequently used pdfs and their factory spec

Gaussian	`Gaussian::g(x,mean,sigma)`
Breit-Wigner	`BreitWigner::bw(x,mean,gamma)`
Landau	`Landau::l(x,mean,sigma)`
Exponential	`Exponential::e(x,alpha)`
Polynomial	`Polynomial::p(x,{a0,a1,a2})`
Chebychev	`Chebychev::p(x,{a0,a1,a2})`
Kernel Estimation	`KeysPdf::k(x,dataSet)`
Poisson	`Poisson::p(x,mu)`
Voigtian ($=BW\otimes G$)	`Voigtian::v(x,mean,gamma,sigma)`

Factory syntax – using expressions

- Customized p.d.f from interpreted expressions

```
w.factory("EXPR::mypdf('sqrt(a*x)+b',x,a,b)") ;
```

- Customized class, compiled and linked on the fly

```
w.factory("CEXPR::mypdf('sqrt(a*x)+b',x,a,b)") ;
```

- re-parametrization of variables (making functions)

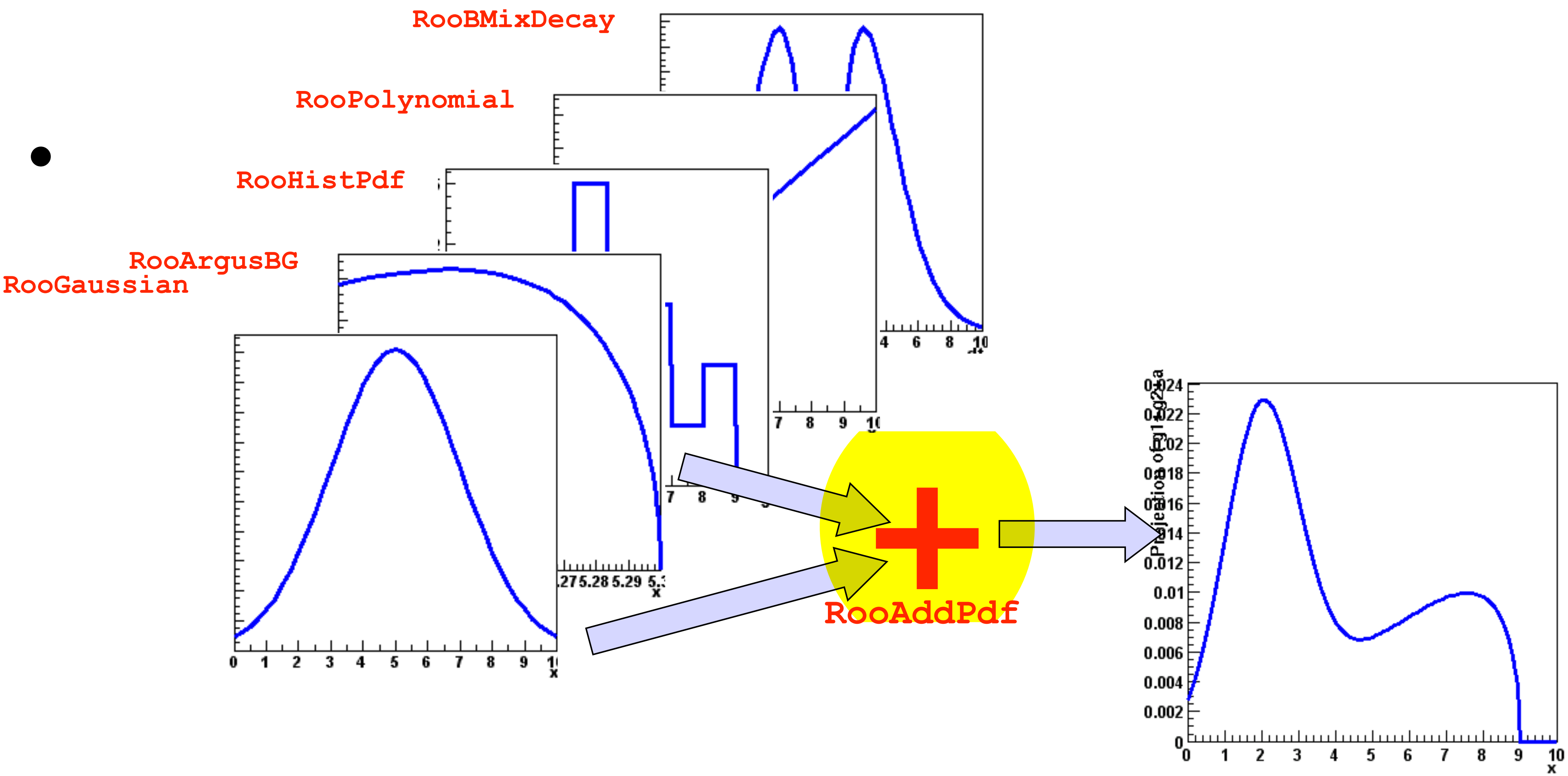
```
w.factory("expr::w('(1-D)/2',D[0,1])") ;
```

  - note using expr (builds a function, a RooAbsReal)
  - instead of EXPR (builds a pdf, a RooAbsPdf)

This usage of upper vs lower case applies also for other factory commands (SUM, PROD,....)

Model building – (Re)using standard components

- Most realistic models are constructed as the sum of one or more p.d.f.s (e.g. signal and background)
- Facilitated through operator p.d.f `RooAddPdf`



Factory syntax: Adding p.d.f.

- Additions of PDF (using fractions)

```
SUM::name(frac1*PDF1,PDFN)
```

```
SUM::name(frac1*PDF1,frac2*PDF2,...,PDFN)
```

  - Note that last PDF does not have an associated fraction

$$F(x) = f \times S(x) + (1 - f)B(x) \quad ; \quad N_{\exp} = N$$

- PDF additions (using expected events instead of fractions)

```
SUM::name(Nsig*SigPDF,Nbkg*BkgPDF)
```

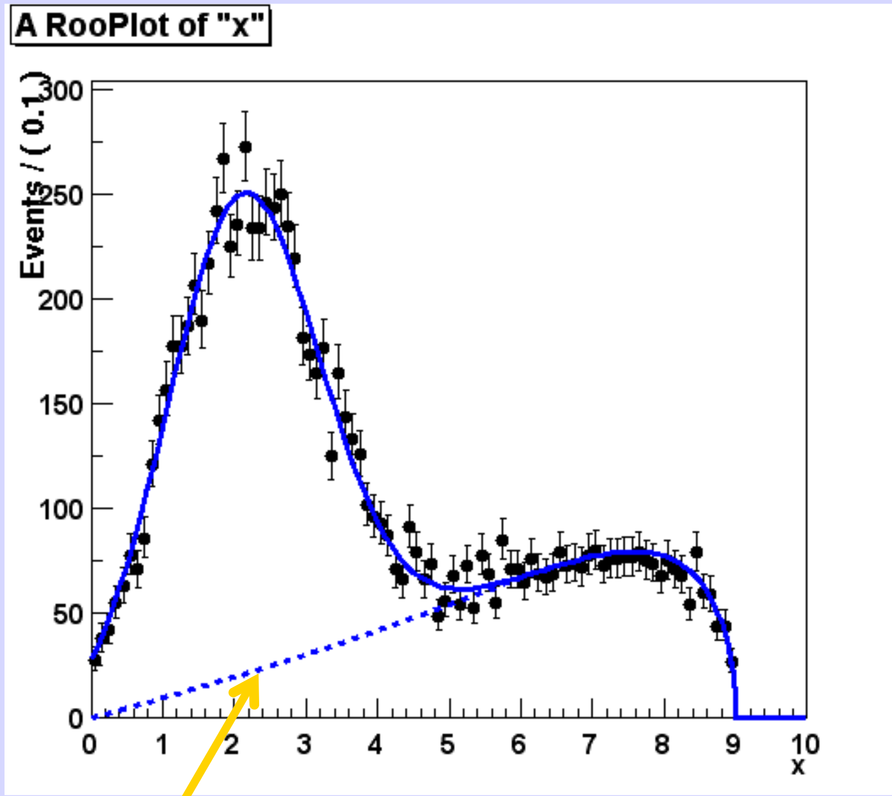
$$F(x) = \frac{N_S}{N_S + N_B} \times S(x) + \frac{N_B}{N_S + N_B} B(x) \quad ; \quad N_{\exp} = N_S + N_B$$

  - the resulting model will be extended
  - the likelihood will contain a Poisson term depending on the total number of expected events (Nsig+Nbkg)

$$L(x \mid p) \to L(x|p)\mathrm{Poisson}(N_{obs}, N_{exp})$$

Component plotting - Introduction

- Plotting, toy event generation and fitting works identically for composite p.d.f.s
  - Several optimizations applied behind the scenes that are specific to composite models (e.g. delegate event generation to components)
- Extra plotting functionality specific to composite pdfs
  - Component plotting

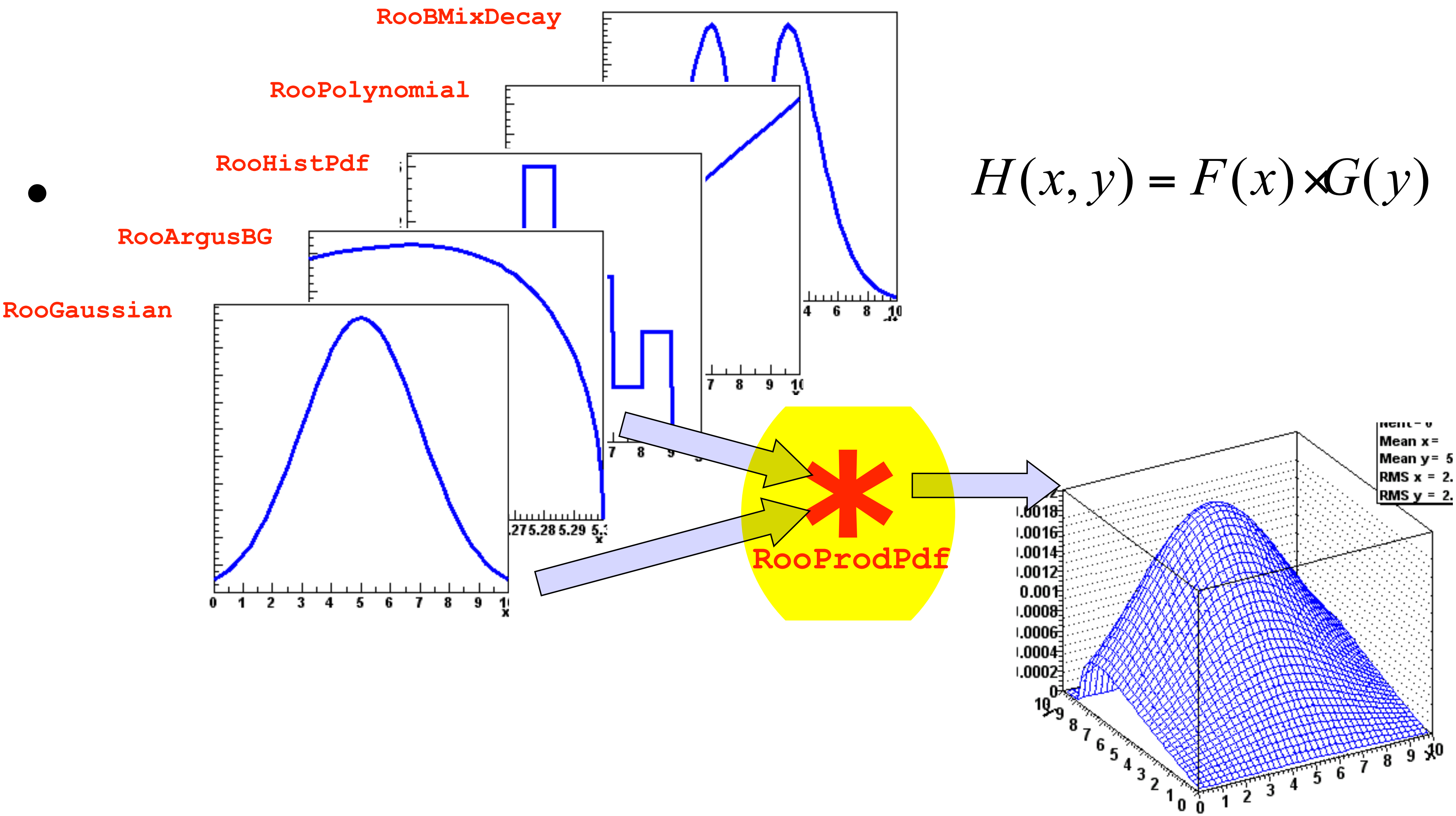



```
// Plot only argus components
w::sum.plotOn(frame,Components("argus"),LineStyle(kDashed)) ;

// Wildcards allowed
w::sum.plotOn(frame,Components("gauss*"),LineStyle(kDashed)) ;
```

Model building – Products of uncorrelated p.d.f.s

$$H(x,y) = F(x) \times G(y)$$



Uncorrelated products – Mathematics and constructors

- Mathematical construction of products of uncorrelated p.d.f.s is straightforward

2D

$$H(x,y) = F(x) \times G(y)$$

nD

$$H(x^{\{i\}}) = \prod_i F^{\{i\}}(x^{\{i\}})$$

  - No explicit normalization required → If input p.d.f.s are unit normalized, product is also unit normalized
  - (Partial) integration and toy MC generation automatically uses factorizing properties of product, e.g. $\int H(x,y)dx \equiv G(y)$ is deduced from structure.

- Corresponding factory operator is PROD

```
w.factory("Gaussian::gx(x[-5,5],mx[2],sx[1])") ;
w.factory("Gaussian::gy(y[-5,5],my[-2],sy[3])") ;

w.factory("PROD::gxy(gx,gy)") ;
```

Introducing correlations through composition

- RooFit pdf building blocks do not require variables as input, just real-valued functions
  - Can substitute any variable with a function expression in parameters and/or observables

$$f(x;p) \Rightarrow f(x,p(y,q)) = f(x,y;q)$$

  - Example: Gaussian with shifting mean

```
w.factory("expr::mean('a*y+b',y[-10,10],a[0.7],b[0.3])") ;
w.factory("Gaussian::g(x[-10,10],mean,sigma[3])") ;
```

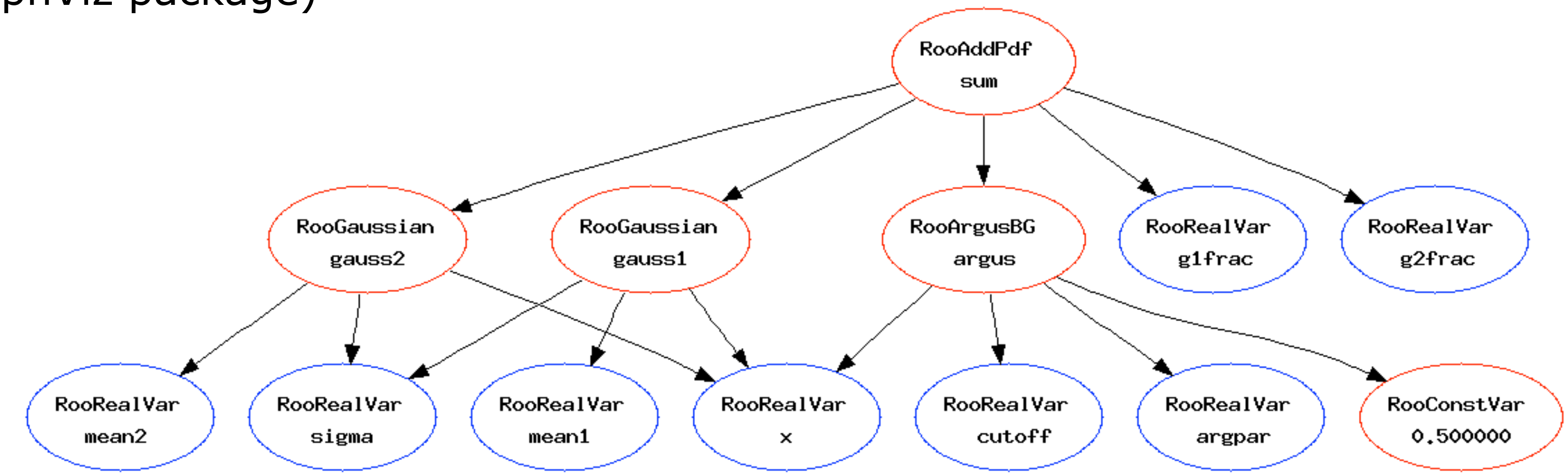
  - No assumption made in function on a,b,x,y being observables or parameters, any combination will work

Operations on specific to composite pdfs

- Tree printing mode of workspace reveals component structure – **w.Print("t")**

```
RooAddPdf::sum[ g1frac * g1 + g2frac * g2 + [%] * argus ] = 0.0687785
    RooGaussian::g1[ x=x mean=mean1 sigma=sigma ] = 0.135335
    RooGaussian::g2[ x=x mean=mean2 sigma=sigma ] = 0.011109
    RooArgusBG::argus[ m=x m0=k c=9 p=0.5 ] = 0
```

  - Can also make input files for GraphViz visualization (**w.pdf("sum")->graphVizTree("myfile.dot")**)
  - Graph output on ROOT Canvas in near future (pending ROOT integration of GraphViz package)



Constructing joint pdfs (RooSimultaneous)

- Operator class SIMUL to construct joint models at the pdf level
  - need a discrete observable (category) to label the channels

```
// Pdfs for channels 'A' and 'B'
w.factory("Gaussian::pdfA(x[-10,10],mean[-10,10],sigma[3])") ;
w.factory("Uniform::pdfB(x)") ;

// Create discrete observable to label channels
w.factory("index[A,B]") ;

// Create joint pdf (RooSimultaneous)
w.factory("SIMUL::joint(index,A=pdfA,B=pdfB)") ;
```

- Construct joint datasets
  - contains observables ("x") and category ("index")

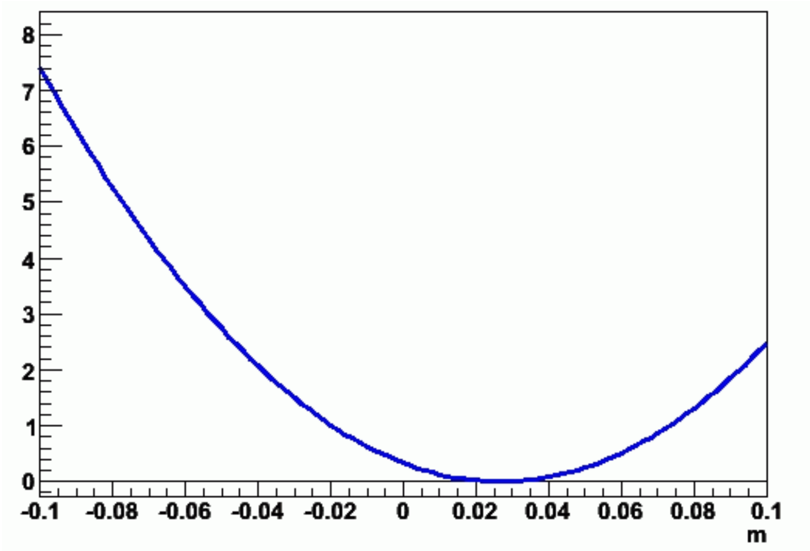
```
RooDataSet *dataA, *dataB ;
RooDataSet dataAB("dataAB","dataAB",
                  RooArgSet(*w.var("x"),*w.cat("index")),
                  Index(*w.cat("index")),
                  Import("A",*dataA),Import("B",*dataB)) ;
```

Constructing the likelihood

- So far focus on construction of pdfs, and basic use for fitting and toy event generation
- Can also explicitly construct the likelihood function of and pdf/ data combination
  - Can use (plot, integrate) likelihood like any RooFit function object

```
RooAbsReal* nll = pdf->createNLL(data) ;

RooPlot* frame = parameter->frame() ;
nll->plotOn(frame,ShiftToZero()) ;
```



Constructing the likelihood

- Example – Manual MIMIZATION using MINUIT
  - Result of minimization are immediately propagated to RooFit variable objects (values and errors)

```
// Create likelihood (calculation parallelized on 8 cores)
RooAbsReal* nll = w::model.createNLL(data,NumCPU(8)) ;


RooMinimizer m(*nll) ;              // create Minimizer class
m.minimize("Minuit2","Migrad");    // minimize using Minuit2
m.hesse() ;                         // Call HESSE
m.minos(w::param) ;                 // Call MINOS for 'param'

RooFitResult* r = m.save() ; // Save status (cov matrix etc)
```

  - Also other minimizers (Minuit, GSL etc) supported

  - N.B. Different minimizer can also be used from RooAbsPdf::fitTo

```
//fit a pdf to a data set using Minuit2 as minimizer
pdf.fitTo(*data, RooFit::Minimizer("Minuit2","Migrad")) ;
```

Basics – Importing data

- Unbinned data can also be imported from ROOT **`TTrees`**

```
// Import unbinned data
RooDataSet data("data","data",x,Import(*myTree)) ;
```

  - Imports `TTree` branch named "x".
  - Can be of type `Double_t`, `Float_t`, `Int_t` or `UInt_t`. All data is converted to Double_t internally
  - Specify a `RooArgSet` of multiple observables to import multiple observables

- Binned data can be imported from ROOT **`THx`** histograms

```
// Import binned data
RooDataHist data("data","data",x,Import(*myTH1)) ;
```

  - Imports values, binning definition *and* SumW2 errors (if defined)
  - Specify a `RooArgList` of observables when importing a TH2/3.

RooFit Summary

- Overview of RooFit functionality
  - not everything covered
  - not discussed on how it works internally (optimizations, analytical deduction, etc..)
- Capable to handle complex model
  - scale to models with large number of parameters
  - being used for many analysis at LHC
- Workspace:
  - easy model creation using the factory syntax
  - tool for storing and sharing models (analysis combination)

RooFit Documentation

- Starting point: http://root.cern.ch/drupal/content/roofit
- Users manual (134 pages ~ 1 year old)
- Quick Start Guide (20 pages, recent)
- Link to 84 tutorial macros (also in $ROOTSYS/tutorials/roofit)
- More than 200 slides from *W. Verkerke* documenting all features are available at the *French School of Statistics 2008*
  - http://indico.in2p3.fr/getFile.py/access?contribId=15&resId=0&materialId=slides&confId=750

Time For Exercises !

Follow the RooFit exercises at the Twiki page:

https://twiki.cern.ch/twiki/bin/view/RooStats/RooStatsTutorialsMarch2015