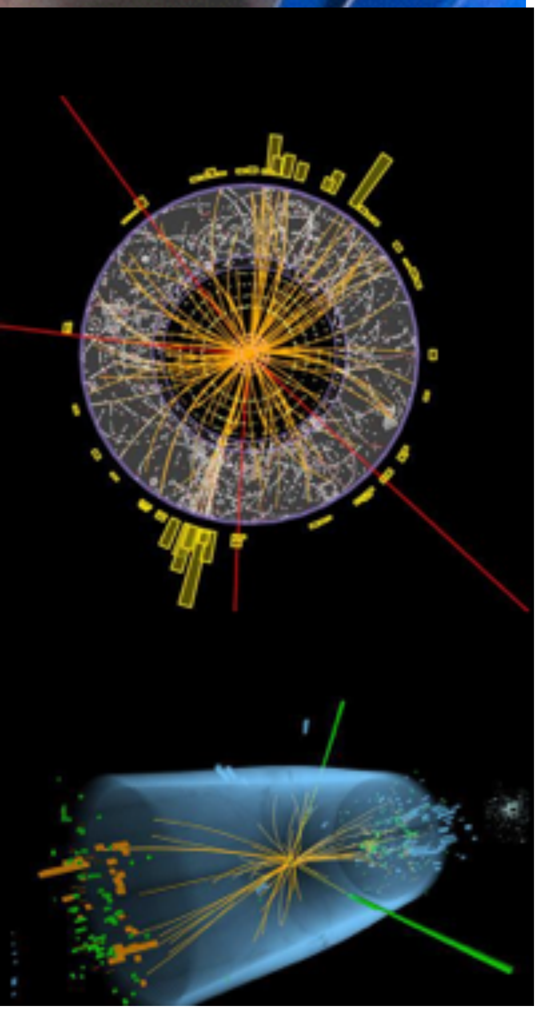
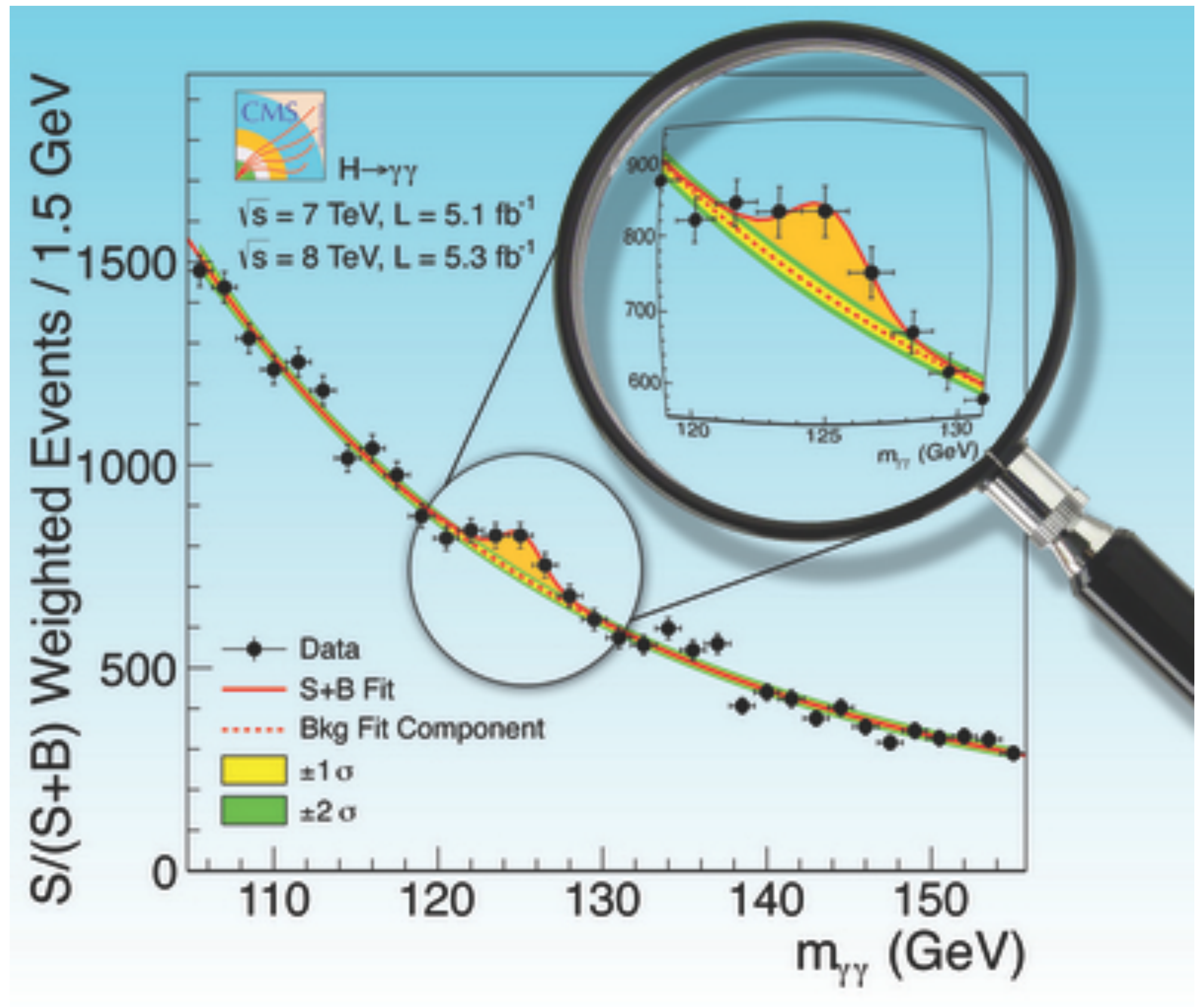


ROOT and Statistic Tutorial at UERJ

4. Fitting and Parameter Estimation



Rio de Janeiro, 24 November 2015

Dr Lorenzo Moneta
CERN PH-SFT
CH-1211 Geneva 23
sftweb.cern.ch
root.cern.ch

Introduction

- We have covered until now
  - Introduction to ROOT
  - Working with histograms in ROOT
  - Data I/O and ROOT Tree

- Introduction of statistics for data analysis
  - Definition of probabilities
  - Frequentist and Bayesian probability
  - Parameter Estimation
  - Introduction to Hypothesis Testing

- Machine Learning
  - Introduction and most popular ML methods

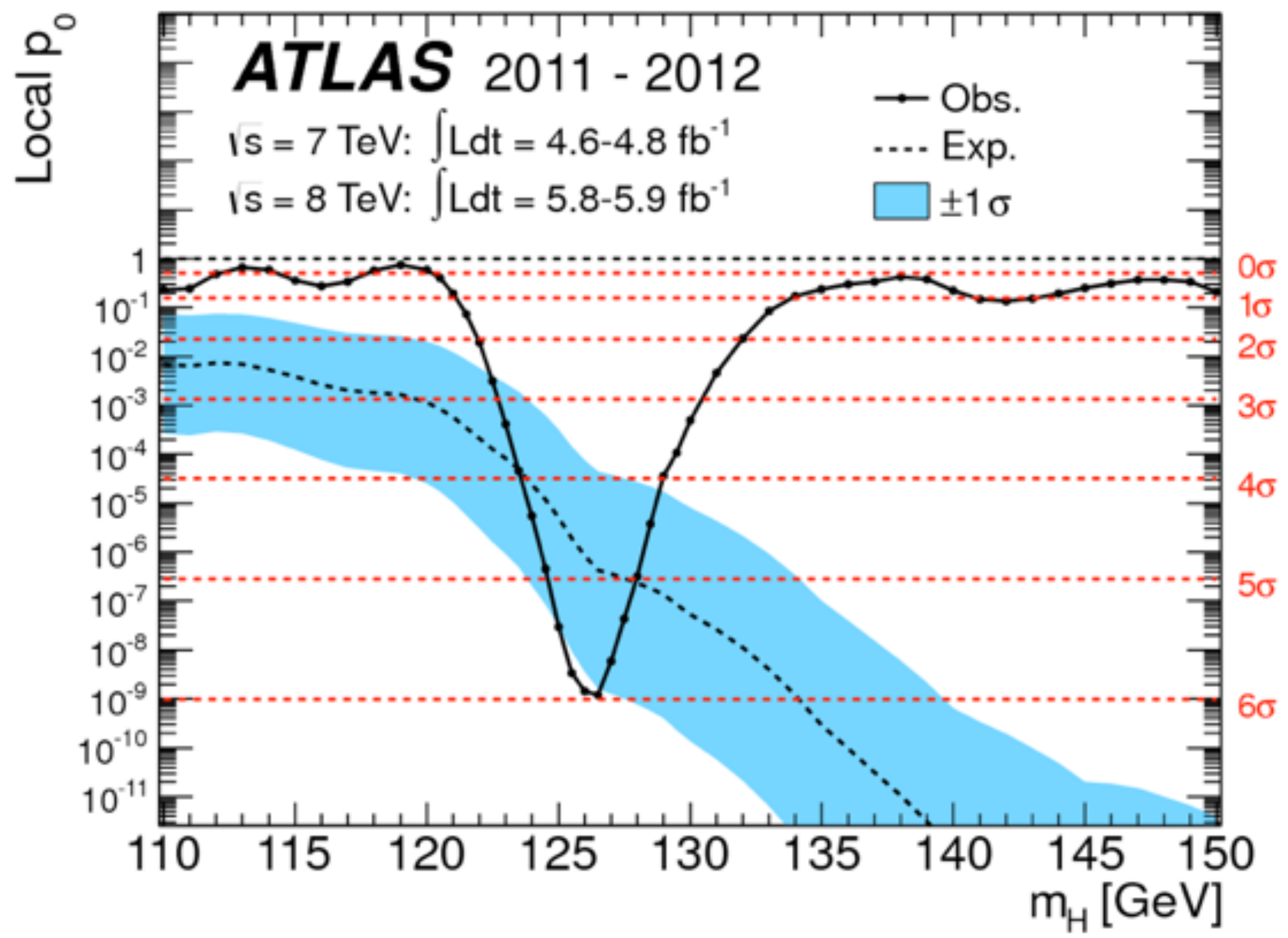
Outline for this lecture

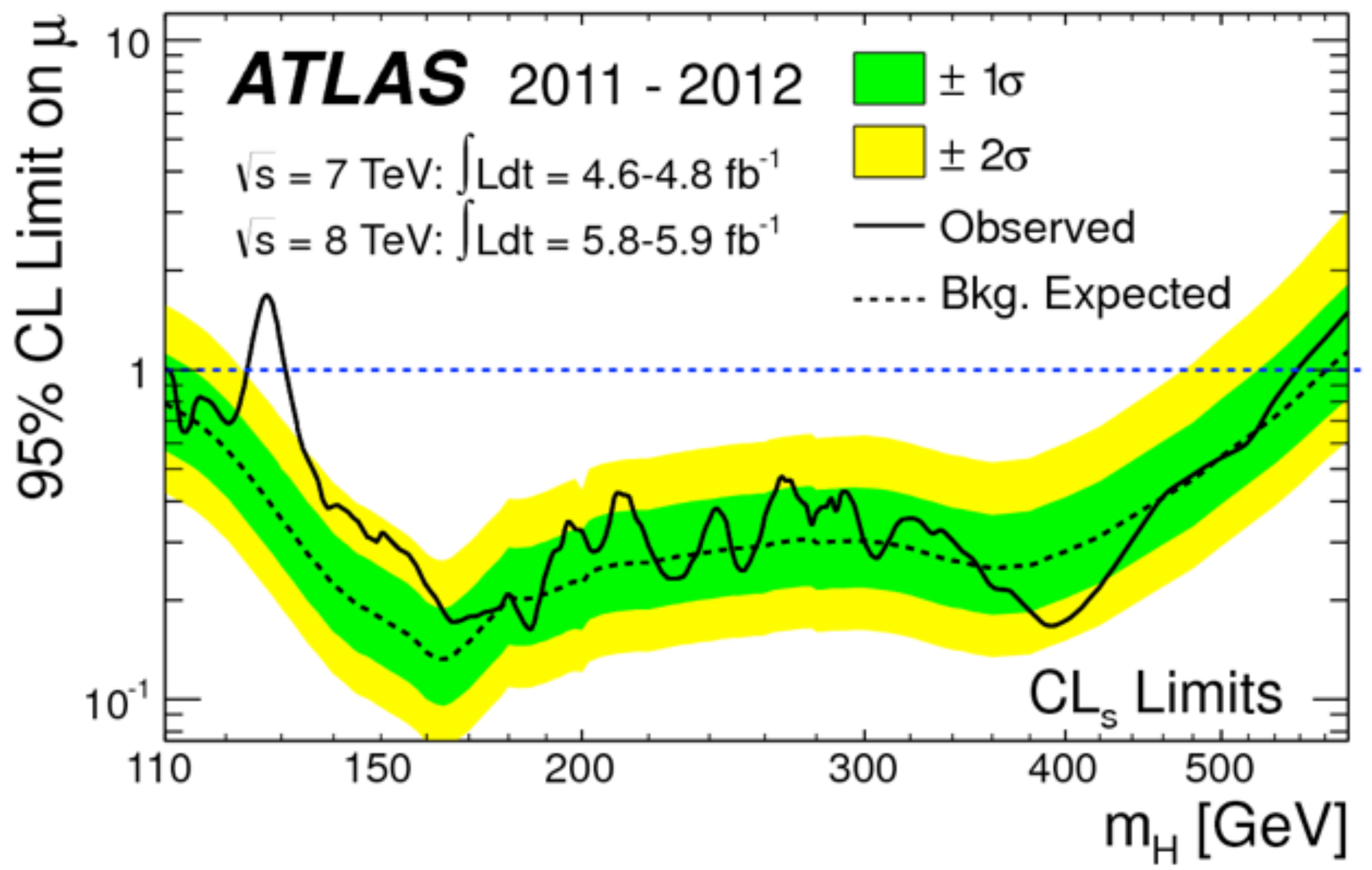
- Recap on theory of parameter estimation
- See its practical applications
  - fitting data points and histograms
- Fitting in ROOT
  - show some examples (e.g using IPhython notebooks)
- Determination of Parameter uncertainties
- Minimisation

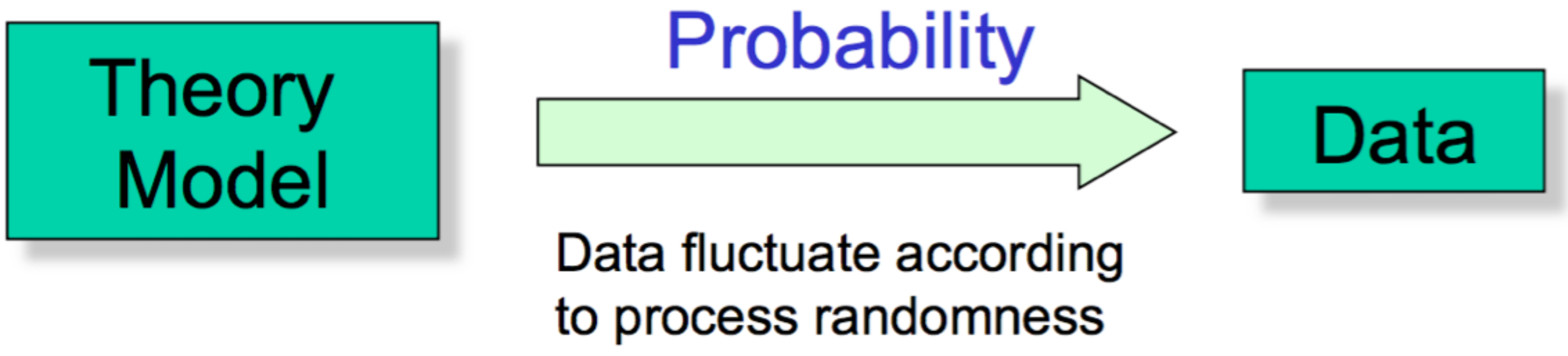
- Introduce RooFit
- How to build complex models for fitting
- Examples of usage

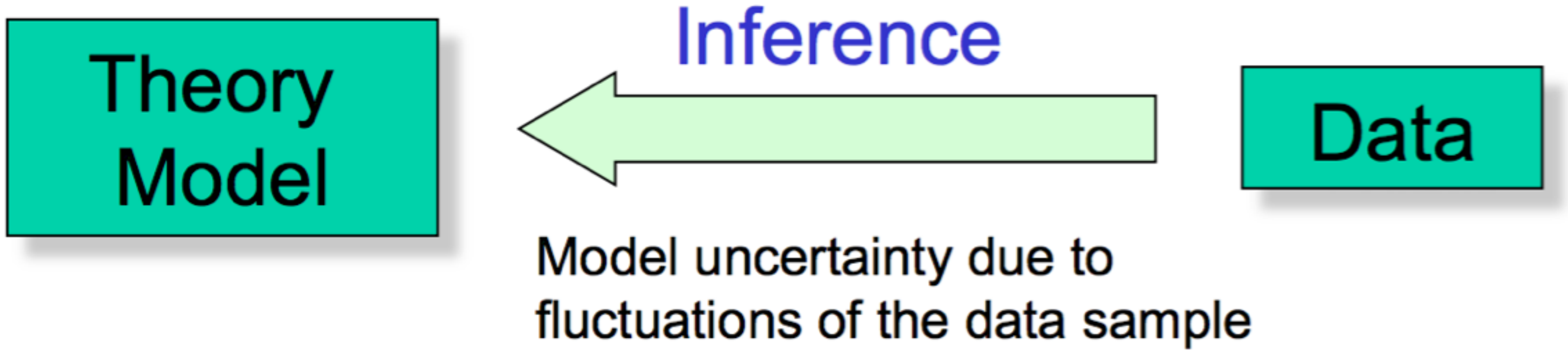
Next Lecture

- Understand better confidence intervals and hypothesis testing
- See practical examples of estimating frequentist and bayesian intervals using RooStats
  - e.g. show how to make Brazilian plots with RooStat
- See examples of estimating discovery significance

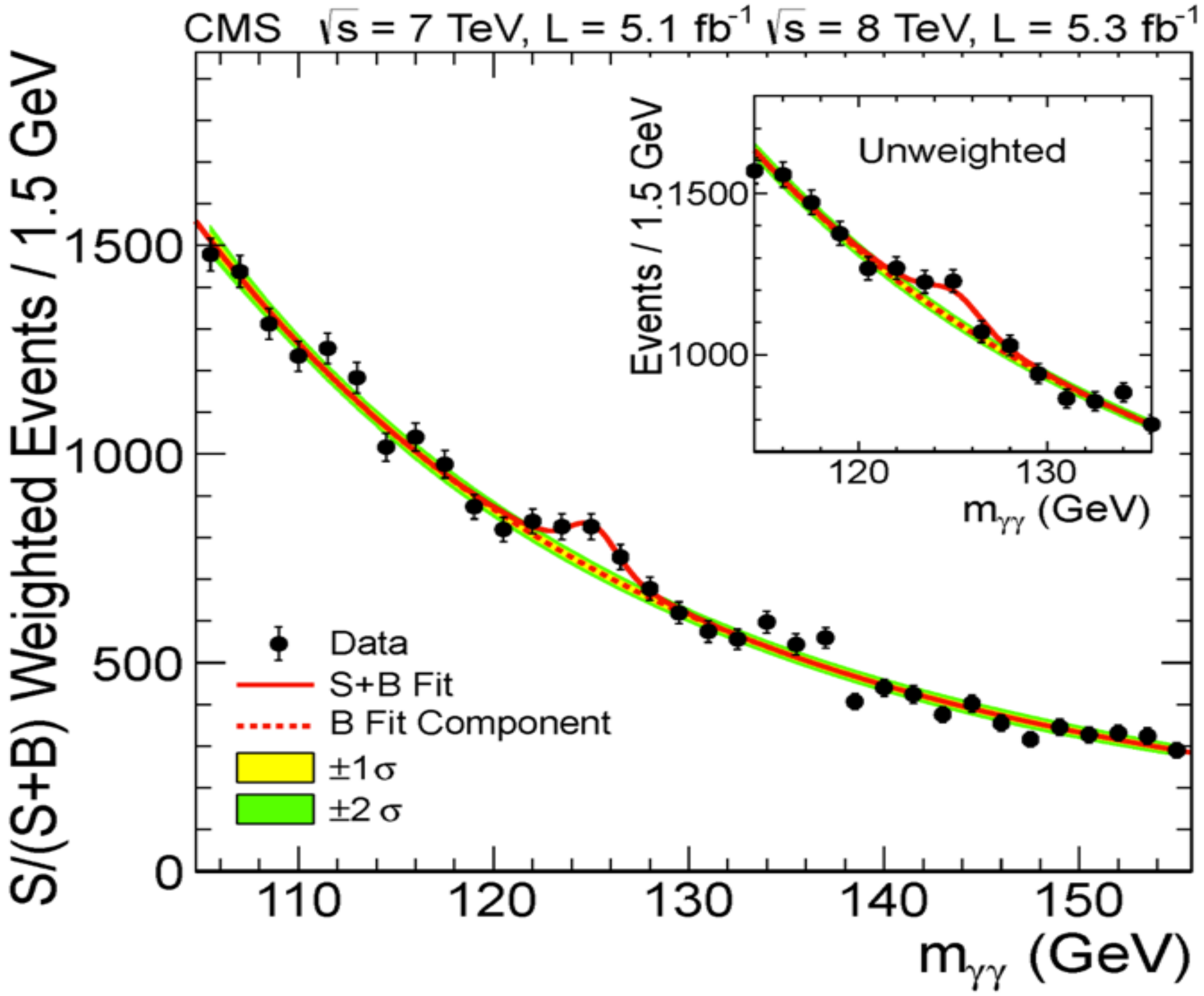





Theory Model
Probability
Data
Data fluctuate according to process randomness


Theory Model
Inference
Data
Model uncertainty due to fluctuations of the data sample

- What is Fitting ?
  - It is the process used to estimate parameters of an hypothetical distribution from the observed data distribution



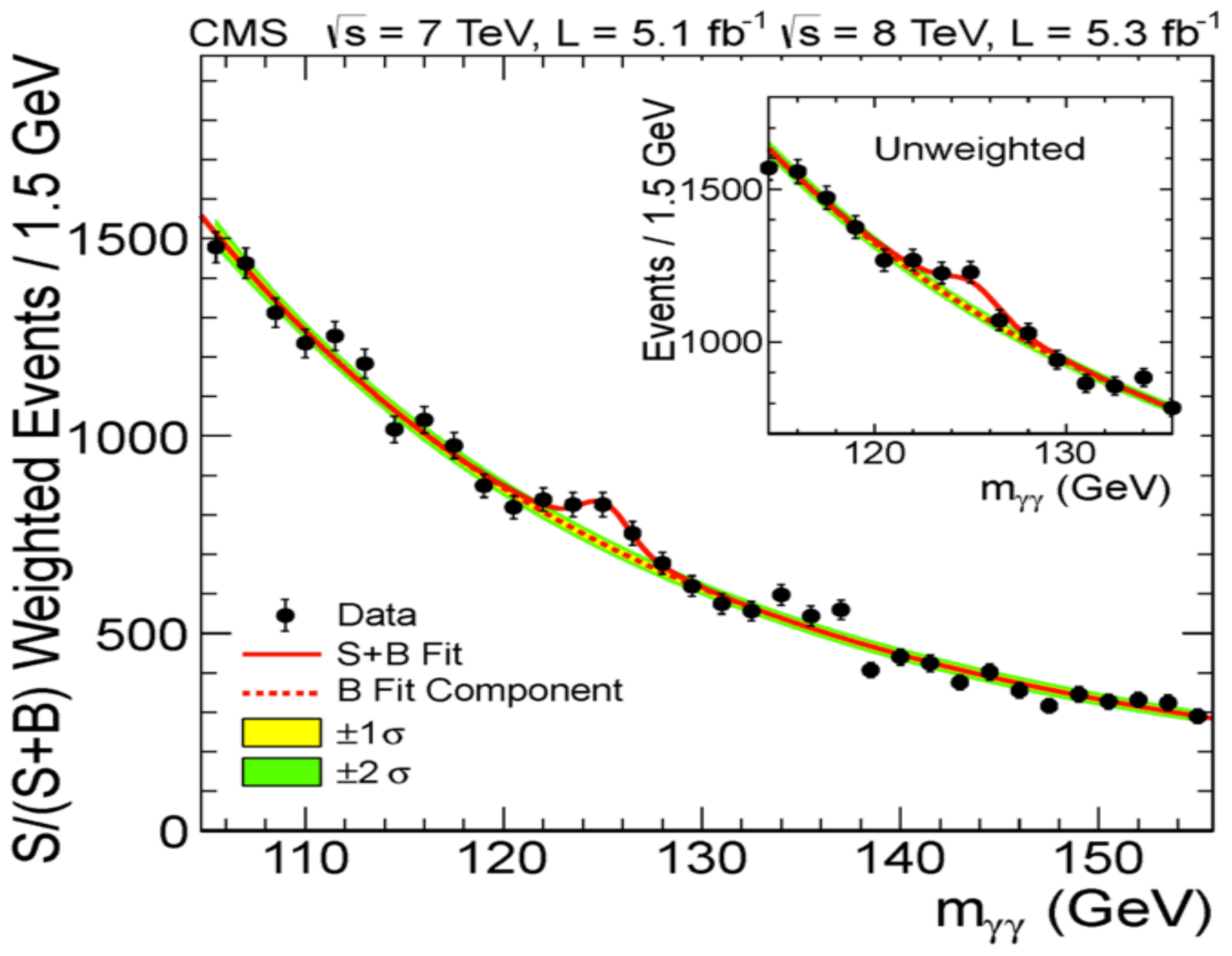
Example

Higgs search in CMS ($H \rightarrow \gamma\gamma$)

We fit for the expected number of Higgs events and for the Higgs mass

Why ?

- One perform fits for:
  - estimate parameter values from a model
    - e.g. location of a resonance in a spectrum or its width
  - Test hypothesis
    - e.g. test the significance of a peak
    - Example: Higgs search in CMS (H → γγ)



Recap on Parameter Estimation

- Given a model for our observed data (Probability Density Function) we want to estimate the parameter of our model
- The model of the observed data is expressed using the Probability Density Function (PDF)
  - the PDF is a differential probability $f(\overrightarrow{x},\theta)$
    - e.g. probability of observing event in an histogram bin $P_{bin} = \int_{bin} f(\overrightarrow{x},\theta) d\overrightarrow{x}$
  - the PDF is normalised to 1 when integrated in all the sample space Ω $\int_{\Omega} f(\overrightarrow{x},\theta) d\overrightarrow{x} = 1$
- To estimate the parameter we use the Likelihood Function

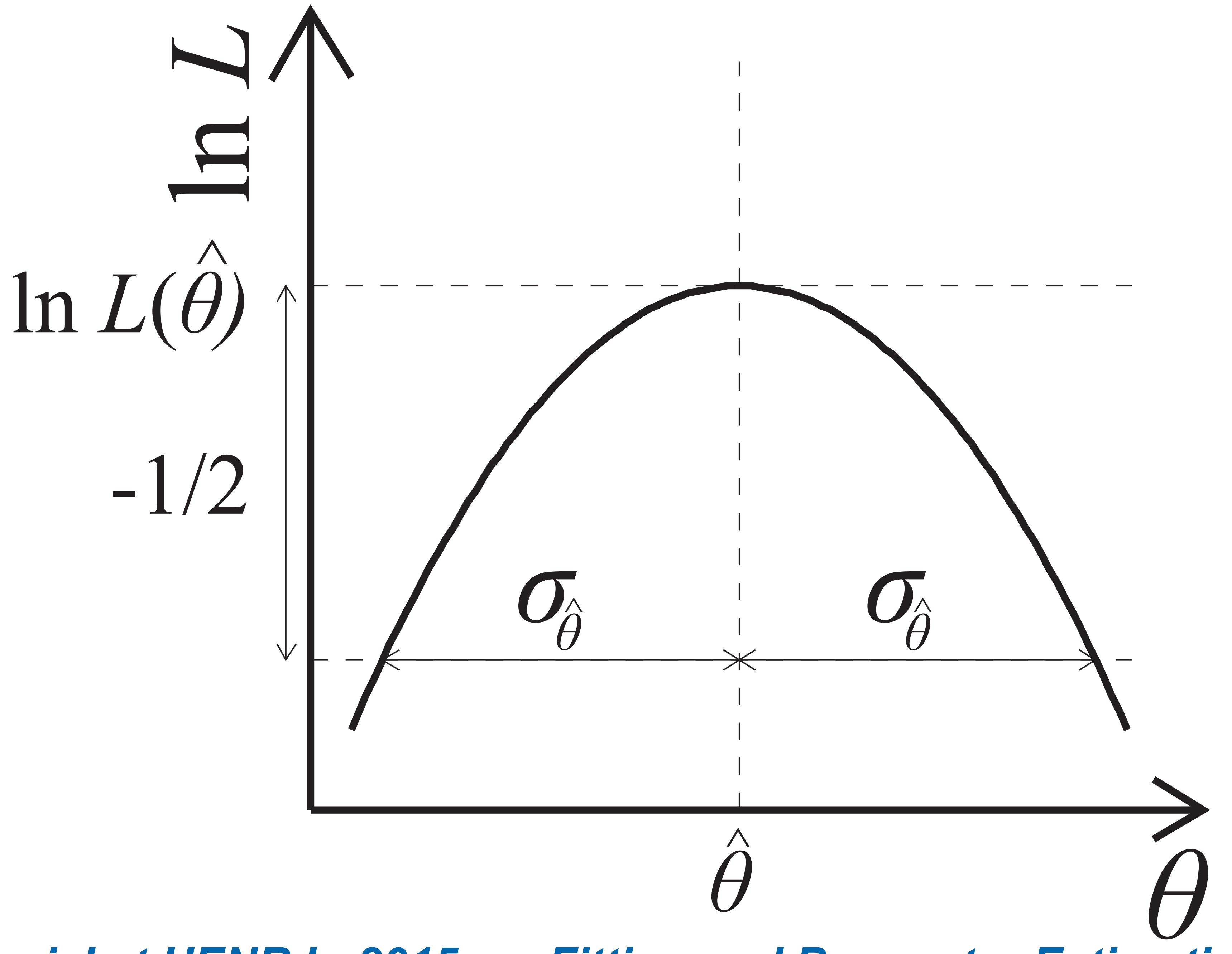
$$L(\overrightarrow{x}_1, ..., \overrightarrow{x}_N | \theta) = \prod_{i=1}^{N} f(\overrightarrow{x}_i, \theta)$$

Maximum Likelihood Estimator

- The ML estimate of the parameter are those who maximise the likelihood function

$$L(\overrightarrow{x}_1, ..., \overrightarrow{x}_N|\theta) = \prod_{i=1}^{N} f(\overrightarrow{x}_i, \theta)$$

$$\text{Best Estimate } \hat{\theta} \longleftarrow \text{Max}(L(x|\theta))$$



ML is the preferred estimator given its good properties:
- consistent
- asymptotically unbiased
- efficient

Maximum Likelihood Solution

- More convenient to work with the log of the likelihood-function
- Use negative log-likelihood function and find global minimum

$$-\log L(\overrightarrow{x}|\theta) = -\sum_i \log f(\overrightarrow{x}_i|\theta)$$

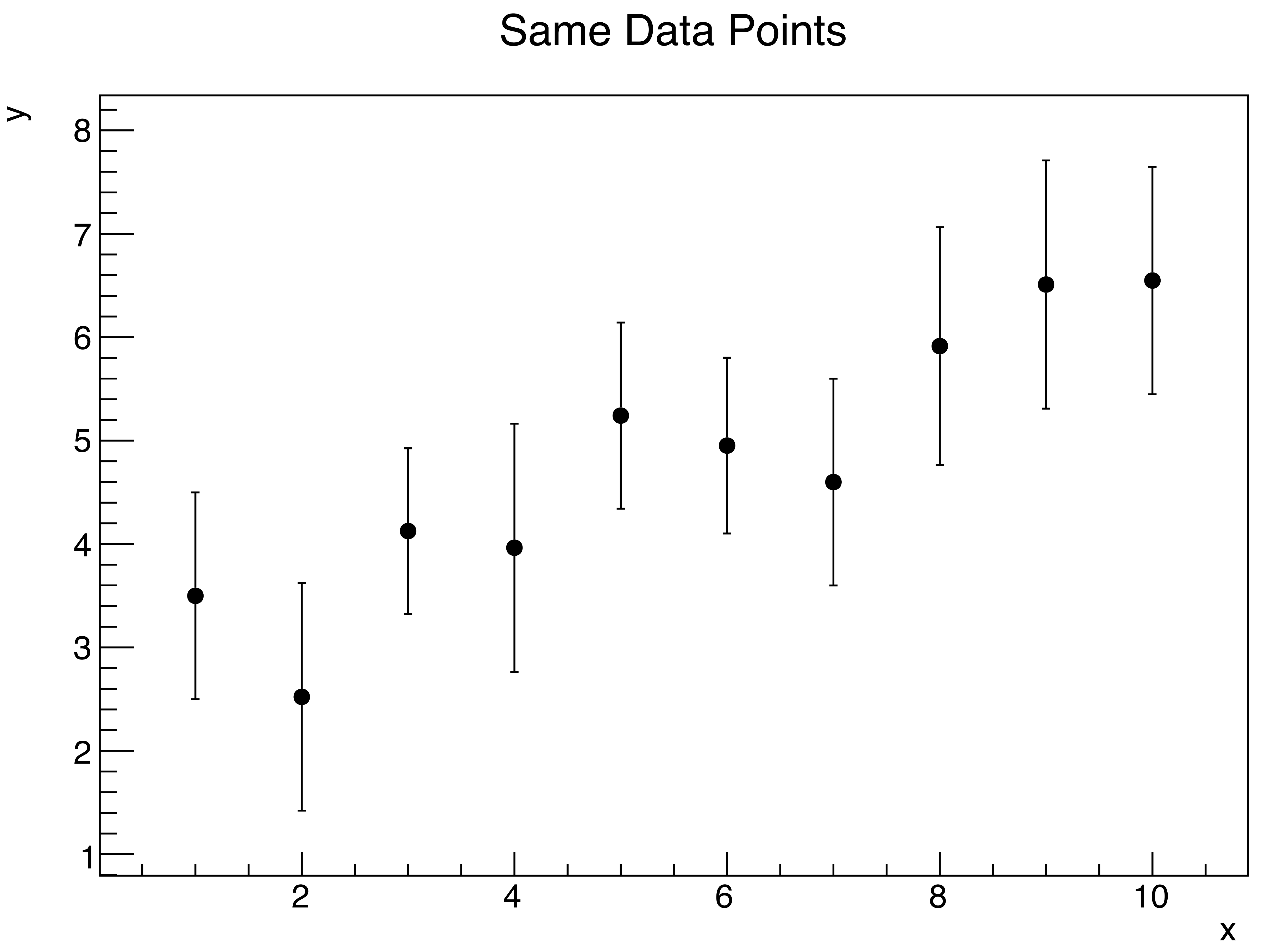
- The PDF must be normalised such that the integral of the likelihood function does not depend on the parameters θ

$$\int_\Omega f(\overrightarrow{x},\theta)d\overrightarrow{x} = 1$$

- The minimum is found typically using a numerical procedure
  - e.g. program MINUIT

Example Fitting Data Points

- We have some data points



Example Fitting Data Points (2)

- Model
  - `y = A * x + B`
- What is the PDF for the observed values (y_1,..y_N) ?

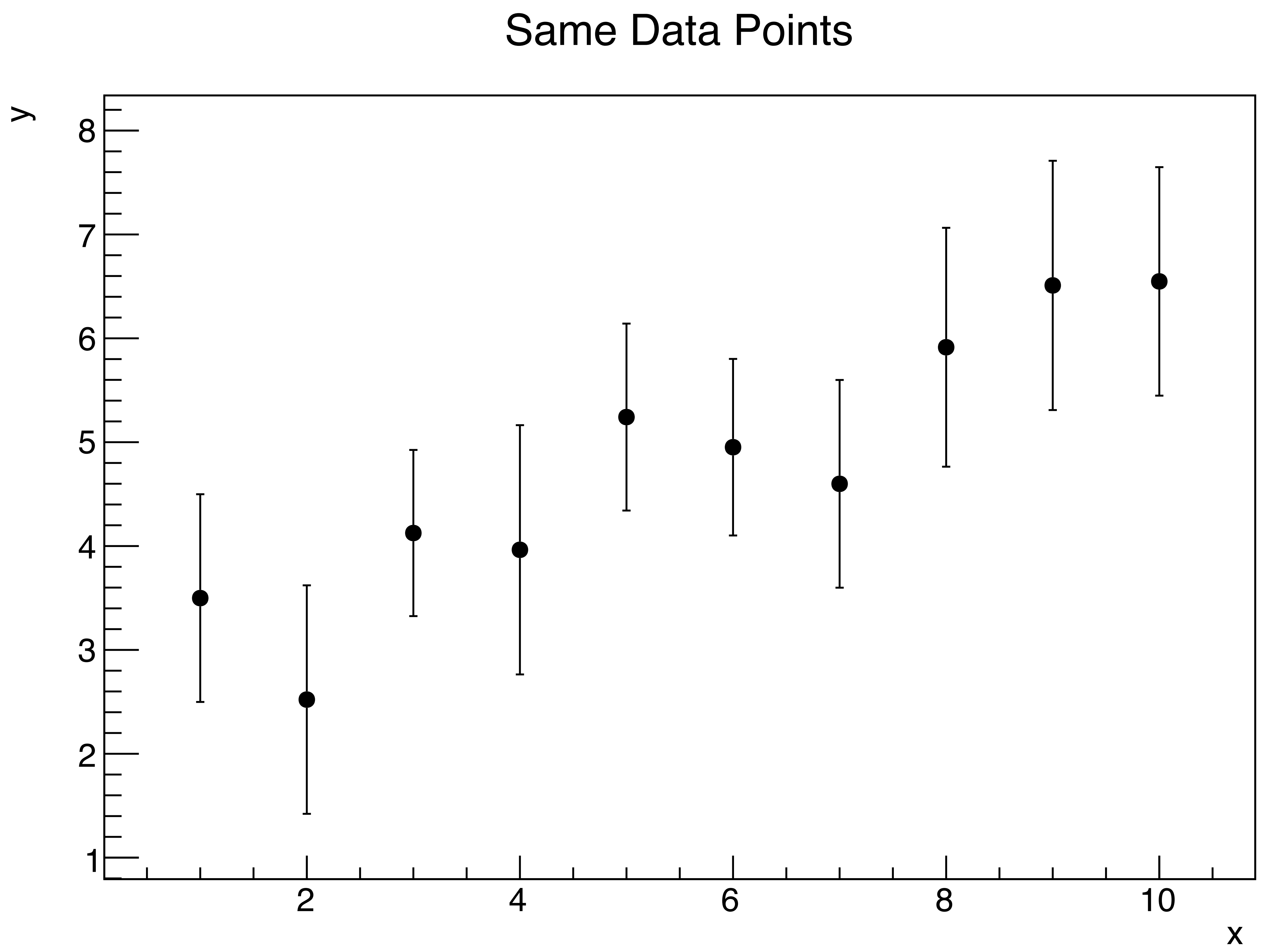
- We assume a normal distribution

$$\mathrm{Gauss}(y_i, y_{\mathrm{exp}}, \sigma) = G(y_i, A * x_i + B, \sigma_i)$$

- Likelihood function

$$L(y_1, ..., y_N | A, B) = \prod_{i=1}^{N} G(y_i, A * x_i + B, \sigma_i)$$

We assume the point error, σ_i , are known



Likelihood for Gaussian points

- The negative log-likelihood function is in this case equivalent to the least-square function ($\chi 2$)

$$\log L(y|\theta) = \sum_{i=1}^{N} \log G(y_i, f(x_i|\theta), \sigma_i) =$$

$$= \sum_{i=1}^{N} \log \frac{1}{\sqrt{2\pi}\sigma_i} e^{-\frac{(y_i - f(x_i|\theta))^2}{2\sigma_i^2}}$$

$$= -\frac{1}{2} \sum_{i=1}^{N} \left(\frac{y_i - f(x_i|\theta)}{\sigma_i} \right)^2$$

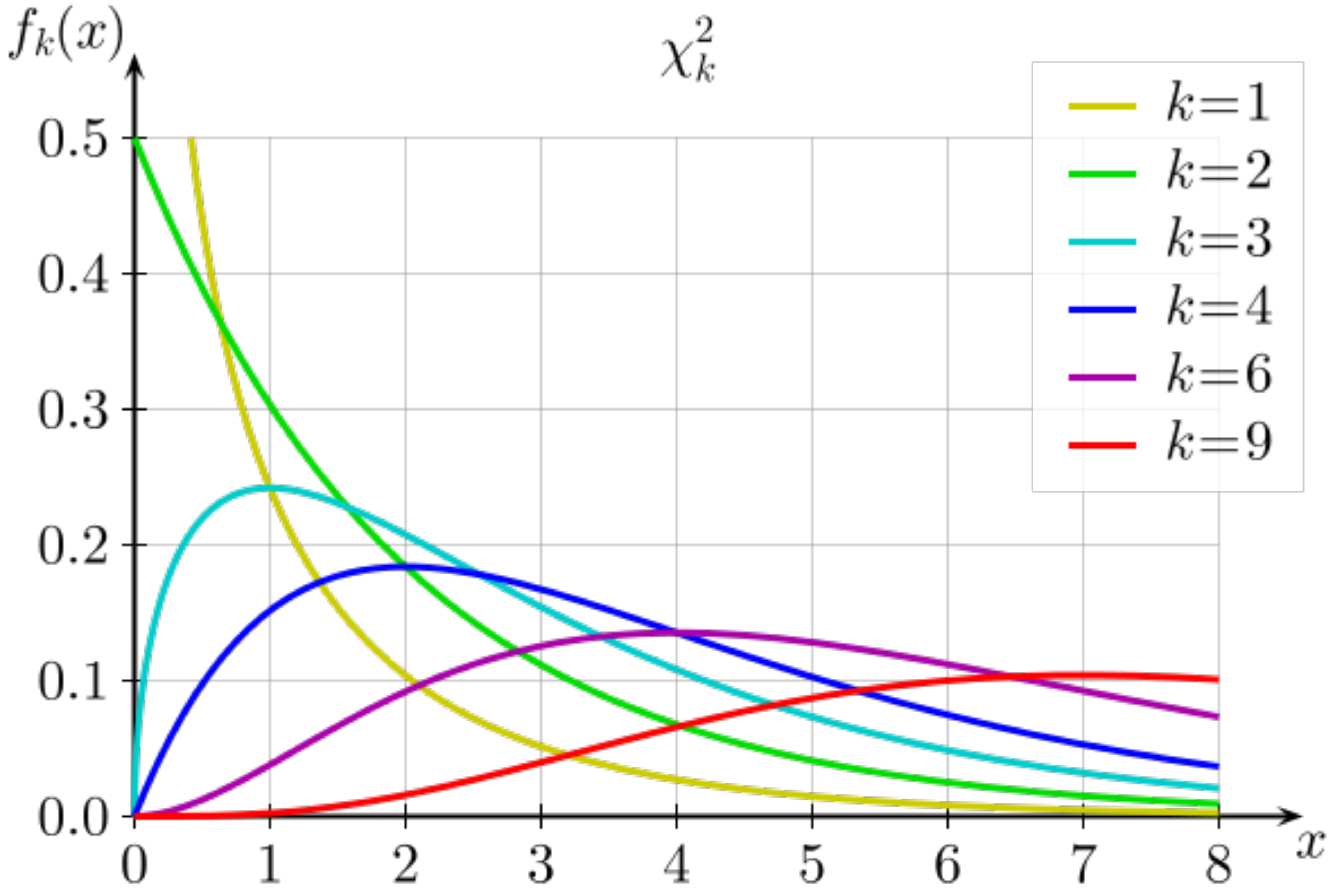
$$-2 \log L(y|\theta) \equiv \chi^2 = \sum_{i=1}^{N} \left(\frac{y_i - f(x_i|\theta)}{\sigma_i} \right)^2$$

Distribution of least-square function is a $\chi 2$ distribution

Chi-squared Distribution

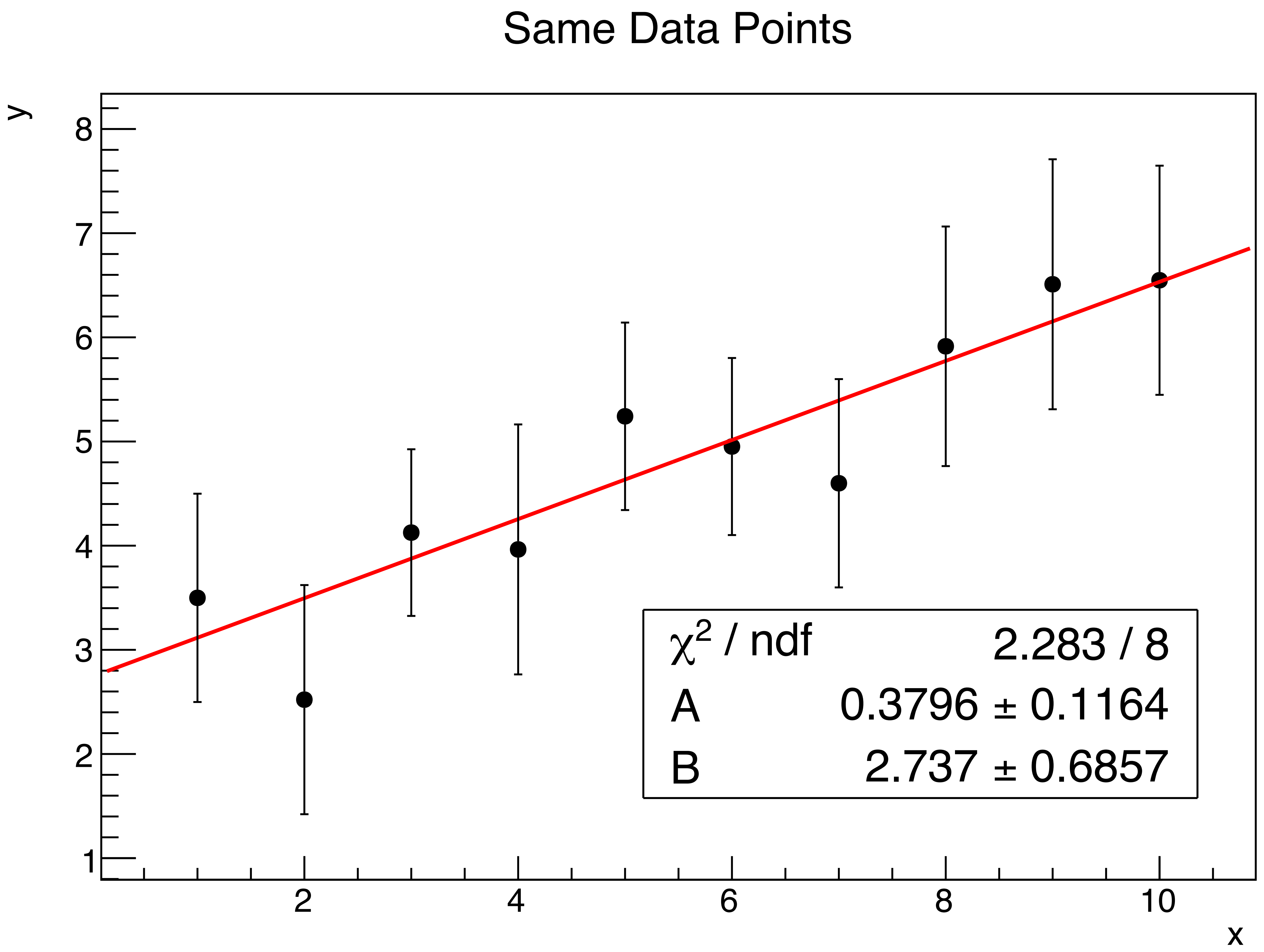
- Distribution for the sum of squared of independent standard normal distributions
  - $z_1,\ldots.z_N$: N variables that are normal distributed $\mathcal{N}(0,1)$
  - $$Q = \sum_{i=1}^{N} z_i^2$$ is distributed as a chi-squared with N degree of freedom
  - $$Q \sim \chi^2(N)$$
  - $\chi2$ PDF: (k is degree of freedom)

$$f(x;\,k) = \begin{cases} \frac{x^{(k/2-1)}e^{-x/2}}{2^{k/2}\Gamma\left(\frac{k}{2}\right)}, & x > 0; \\ 0, & \text{otherwise.} \end{cases}$$



Fitting Data Points

- Minimize the $\chi 2$ function to find best values of parameters (e.g. A and B)



- For linear functions the solution (minimum) can be found analytically

Recap on Fitting

- A histogram or a graph (set of data points) represents an estimate of an underlying distribution (or a function).
- The data can be used to infer the parameters describing the underlying distribution.
- Assume a relation between the observed variables y and x:
  - $y = f(x \mid \theta)$
    - $f(x \mid \theta)$ is the fit (model) function
    - for an histogram y is the bin content

- Least square fit (χ^2) :
  - minimizes the deviations between the observed y and the predicted function values:
    - weighted by the data point errors
      - $\sigma = \sqrt{N}$ for the histograms

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

  - Equivalent to ML method if the data point distribution is Gaussian

ML Fit of Histogram

- Distribution for the bin content of an histogram is normally Poisson
  - bin records counts, i.e number of events n_{obs}
  - `Poisson( nobs | nexp )`
    - n_{exp} is the expected bin content

$$n_{exp} = N_{TOT} \int_{bin} f(x,\theta)dx \approx N_{TOT} \Delta_x f(x_c|\theta)$$

- Log-Likelihood function is

$$\log L(x|\theta) = \sum_{bin} \log \left(\text{Poisson} \left(n_{obs}^{bin} | f(x_c^{bin}|\theta) \right) \right)$$

$$= \sum_{bin} n_{obs}^{bin} \log f(x_c^{bin}|\theta) - f(x_c^{bin}|\theta) + \text{constant}$$

$$\text{Poisson}\,(n|\nu) = \frac{\nu^n}{n!} e^{-\nu}$$

- Likelihood fit is the correct one for histogram
  - Least square is just an approximation when Poisson → Gaussian ($\sigma = \sqrt{n}$)
- For functions varying a lot within the bin, more correct to use the integral of the model function in the bin

Least-Square Histogram Fit

- Often used least-square fit for histograms

$$\chi^2 = \sum_i \frac{(y_i - f(x_i^c, \theta))^2}{\sigma_i^2}$$

– use observed counts to estimate the bin error $\sigma = \sqrt{n_{obs}}$ (Newman χ2)
  - problem with histogram bins which are empty
    – e.g. ROOT decides to not use such bins in the fit
    – under-estimation of tails
  - This is the default fitting method in ROOT

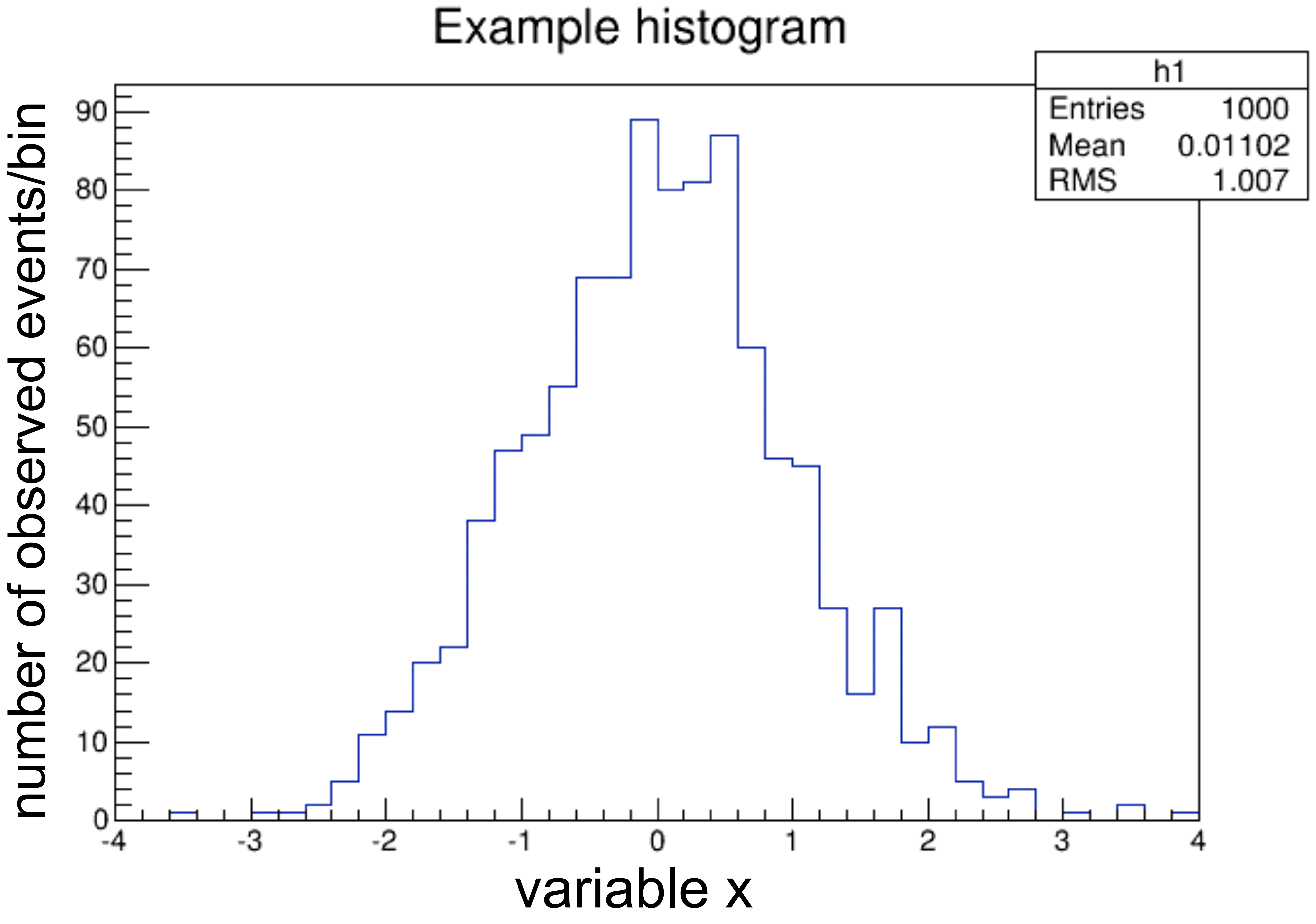
– use expected bin errors : $\sigma = \sqrt{n_{exp}}$ (Pearson χ2)
  - over-estimation of tails
  - error for low-statistics bins is far too small since distribution is not Gaussian !

Fitting in ROOT

- How do we do fit in ROOT:
  - –Create first a parametric function object, `TF1`, which represents our model, *i.e.* the fit function.
  - –Set the initial values of the function parameters.
  - –Fit the data object (Histogram or Graph):
    - call the `Fit` method on the Histogram or Graphs passing the function object as parameter
      - –various options are possibles (see the `TH1::Fit` documentation)
      - –e.g select type of fit : least-square (default) or likelihood (option “L”)
      - –the resulting fit function is then drawn on top of the Histogram or the Graph.
  - –Examine result:
    - get parameter values;
    - get parameter errors (e.g. their confidence level);
    - get parameter correlation;
    - get fit quality.

Simple Gaussian Fitting

- Let’s suppose we have an histogram:
  - we know probably represents a gaussian distribution
  - we don’t know the true parameter of the distribution
  - we want to estimate the mean and sigma of the hypothetical underlying gaussian distribution.



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, cheb0,…10, crystalball,breitwigner`

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);
```

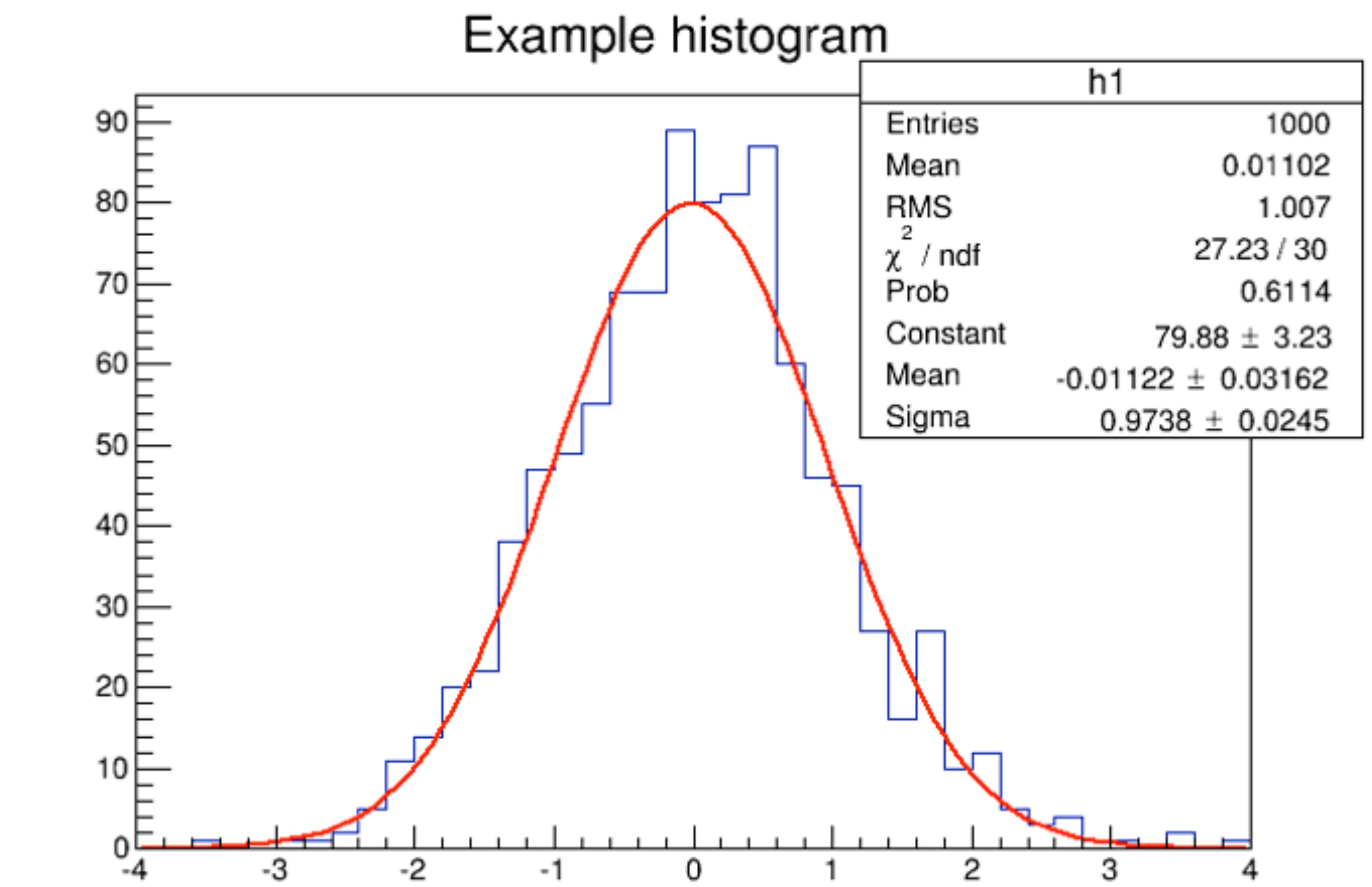
  - e.g using a lambda function (with Cling and C++-11)

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

Fitting Histogram

- How to fit the histogram:
  - after creating the function one needs to set the initial value of the parameters
  - then we can call the `Fit` method of the histogram class

```
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
  NO.   NAME      VALUE            ERROR          SIZE
   1  Constant     7.98760e+01   3.22882e+00   6.64363
   2  Mean        -1.12183e-02   3.16223e-02   8.18642
   3  Sigma        9.73840e-01   2.44738e-02   1.69250
```



For displaying the fit parameters:

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

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 customized C/C++ functions).
- The fit function can be retrieved using its name:

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

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

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

- The parameter errors:

```
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");
```

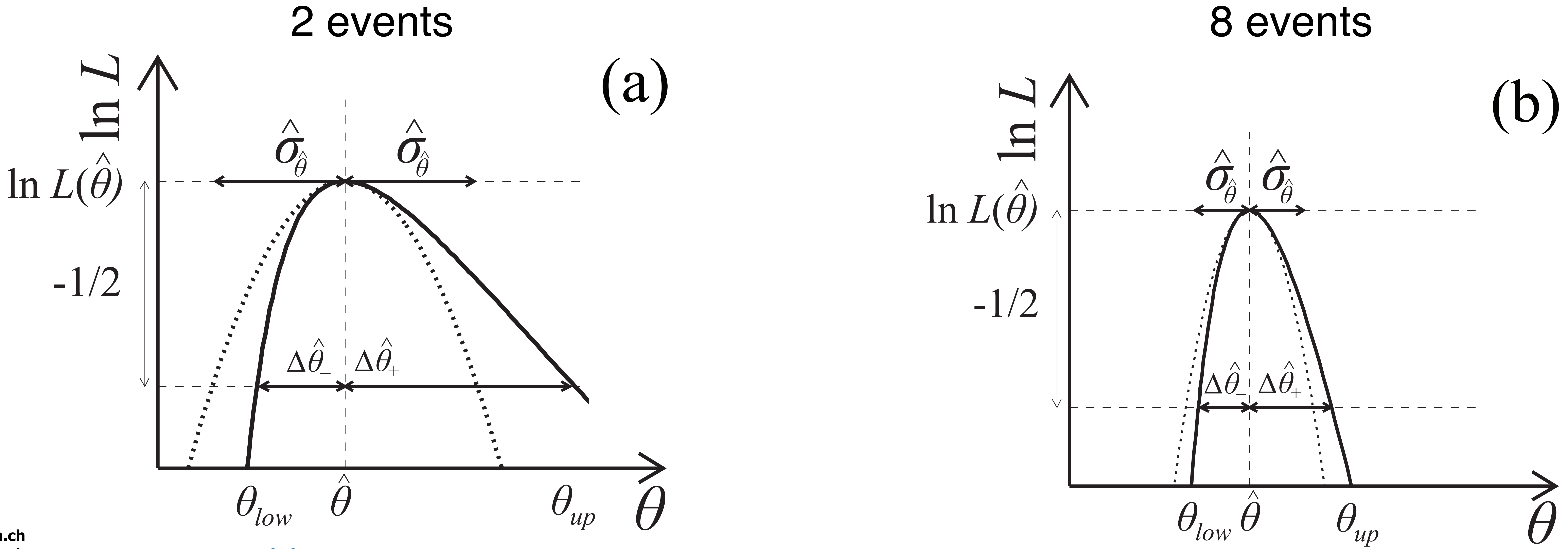
All fitting options documented in reference guide or User Guide (Fitting Histogram chapter)

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:

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

Exponential decay fit



Parameter Errors (2)

- 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.

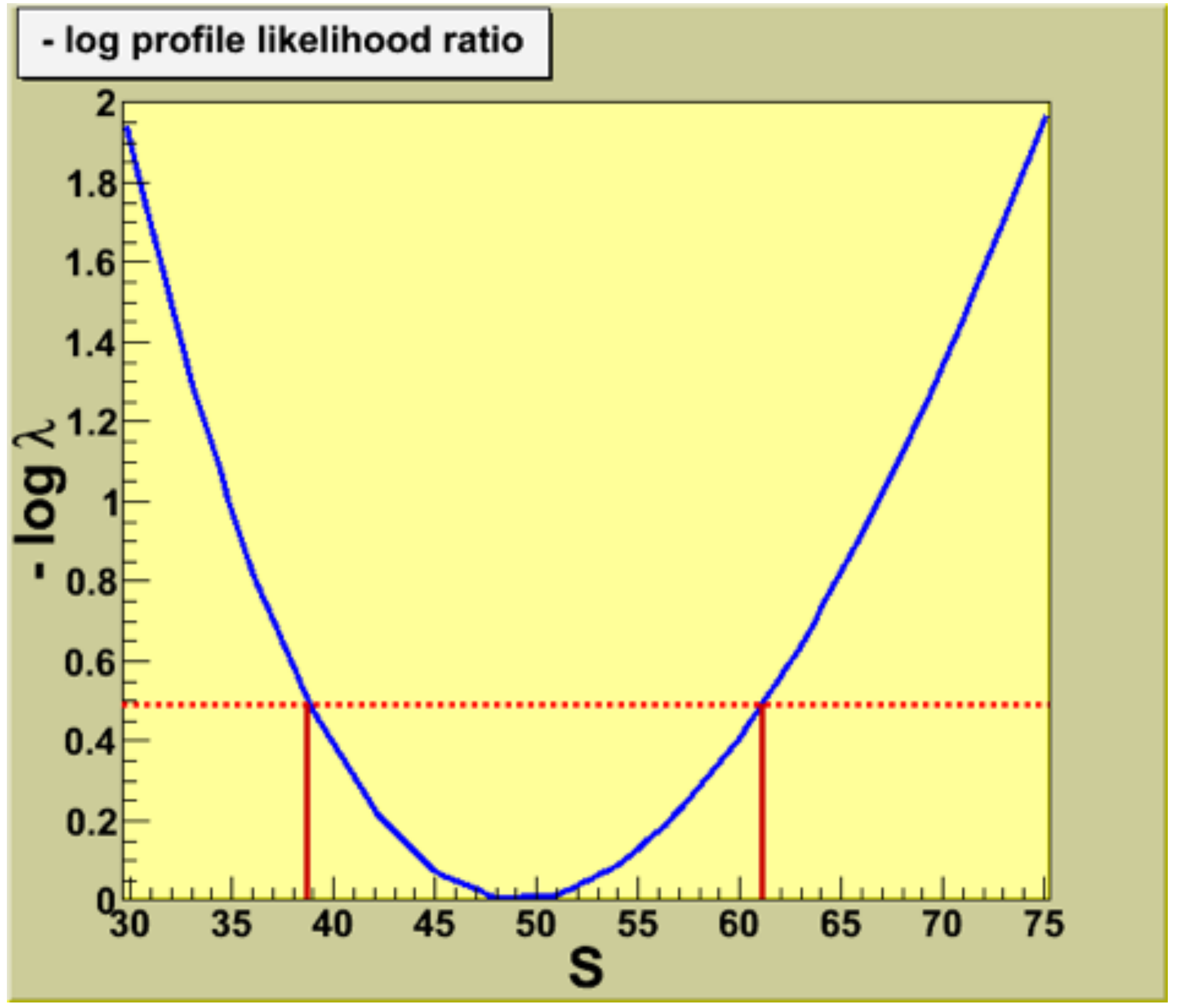
$$\lambda(\theta) = \frac{L(x|\theta)}{L(x|\hat{\theta})} \qquad -2\log\lambda(\theta) \approx (\theta-\hat{\theta})^T H (\theta-\hat{\theta})$$

$$-2\log\lambda(\theta) \sim \chi^2 \text{distribution}$$

$$-\log\lambda(\theta_{low} \equiv \hat{\theta} - \delta\hat{\theta}_-) = -\log\lambda(\theta_{up} \equiv \hat{\theta} + \delta\hat{\theta}_+) = \frac{1}{2}F^{-1}_{\chi^2}(0.68, 1) = 0.5$$

- Method of Minuit/Minos (Fit option “E” in ROOT)
  - obtain a confidence interval which is in general not symmetric around the best parameter estimate

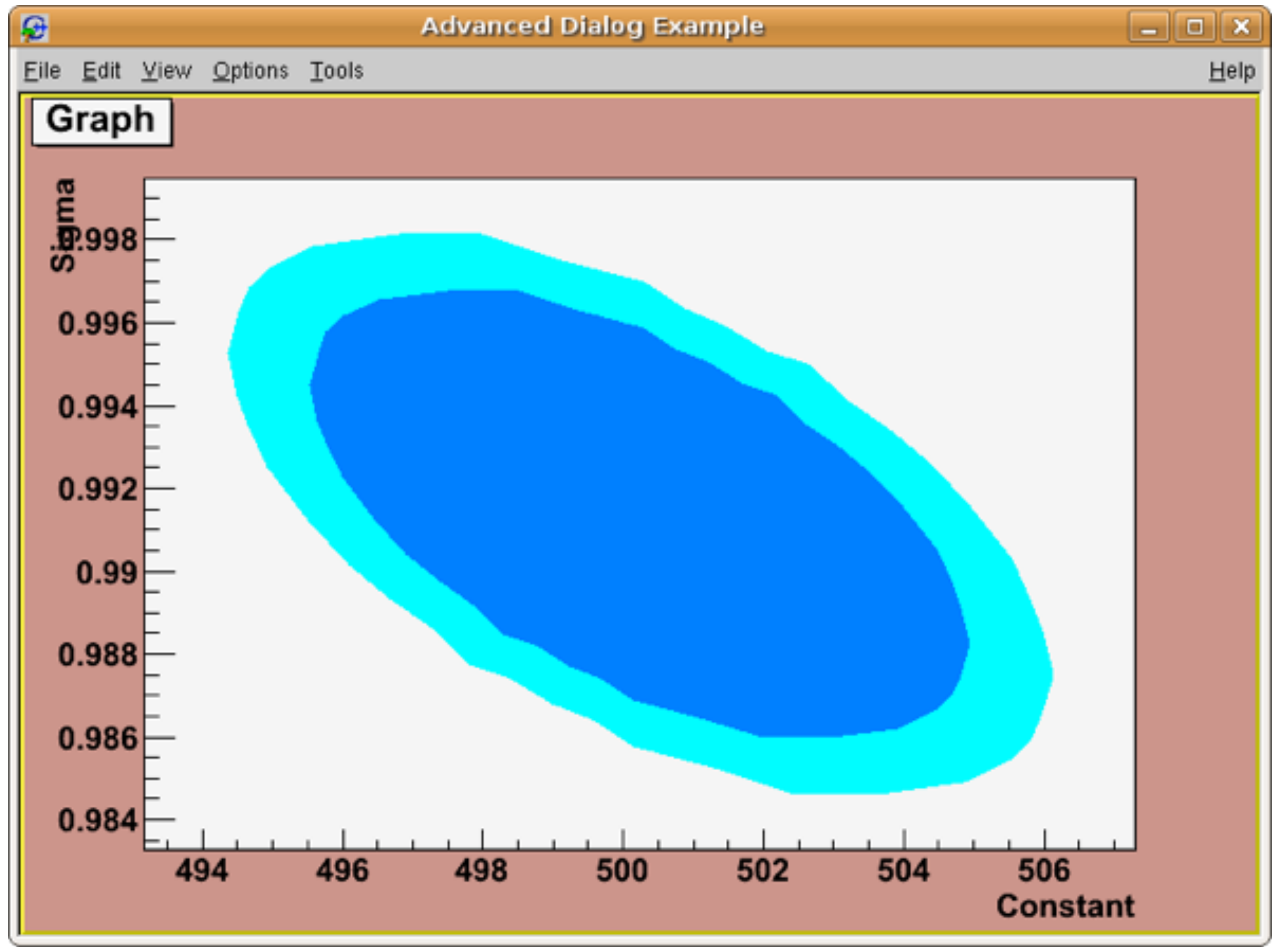
```
TFitResultPtr r = h1->Fit(f1,"E S");
r->LowerError(par_number);
r->UpperError(par_number);
```



Parameter Contours

- In case of more than one parameter of interest one can obtain the contours enclosing the confidence region at a given confidence level (e.g. 68 %)

$$-\log\lambda(\theta_1,\theta_2) = \frac{1}{2}F^{-1}_{\chi^2}(0.68,2) = 1.15$$



Note on Binned Likelihood Fit

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

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

  - -2lnλ(θ) 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 -lnλ(θ)
  - can be retrieved it using `TFitResult::MinFcnValue()`

Extended Likelihood Fit

- **Unbinned likelihood fit**
  - fit each single data point x_i
  - fit only functional shape (no overall normalisation), p.d.f are normalised

$$L(x|\theta) = \prod_{i=1}^{N} f(x_i|\theta)$$

- **Extended likelihood fit**
  - add Poisson fluctuations for observed events
  - fit also the function normalisation (number of expected events)

$$L(x|\theta) = e^{-\nu}\frac{\nu^N}{N!}\prod_{i=1}^{N} f(x_i|\theta)$$

Minimization

- The fitting problem is solved by minimizing the least-square or likelihood function.
- A direct solution exists only in case of linear fitting (function linear in the parameters)
  - e.g fitting polynomials
- Otherwise an iterative numerical algorithm is used:
  - Minuit is the minimization algorithm used by default
    - Two implementations: TMinuit and **Minuit2** (new C++ implementation and recommended)
    - other algorithms exists: Fumili, or minimizers based on GSL, genetic and simulated annealing algorithms
  - To change the minimizer:

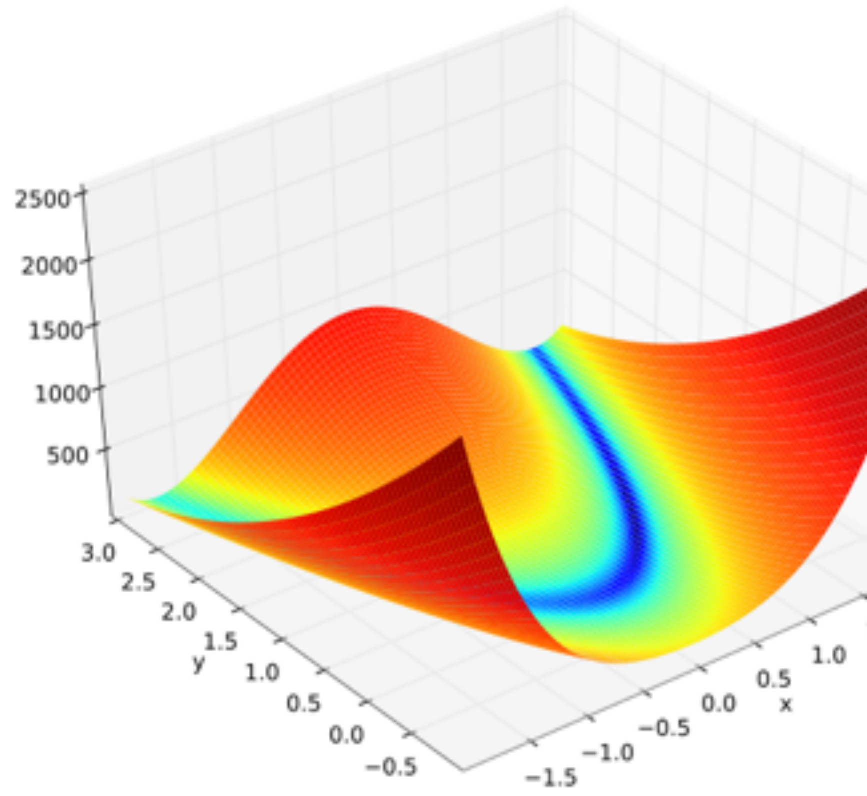
```
ROOT::Math::MinimizerOptions::SetDefaultMinimizer("Minuit2");
```

  - Other commands are also available to control the minimization:
    - e.g. to control tolerance for convergence

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

MINUIT Algorithm

- Migrad based on Variable Metric algorithm (Davidon)
- Iterate to find function minimum:
  - start from initial estimate of gradient $\mathbf{g_0}$ and Hessian matrix, $\mathbf{B_0}$
  - find Newton direction: $\mathbf{d} = \mathbf{B}^{-1}\,\mathbf{g}$
  - computing step by searching for minimum of **F(x)** along **d**
  - compute gradient **g** at the new point
  - update inverse Hessian matrix, $\mathbf{B}^{-1}$ at the new point using an approximate formula (Davidon, Powell, Fletcher)
  - repeat iteration until expected distance from minimum (edm) smaller than required tolerance ($\mathbf{edm} = \mathbf{g}^T\mathbf{B}^{-1}\,\mathbf{g}$)



$$f(x_k + \Delta x) \approx f(x_k) + \nabla f(x_k)^T \Delta x + \frac{1}{2}\Delta x^T B\,\Delta x,$$

$$\nabla f(x_k + \Delta x) \approx \nabla f(x_k) + B\,\Delta x$$

Newton step is obtained by setting this gradient to zero

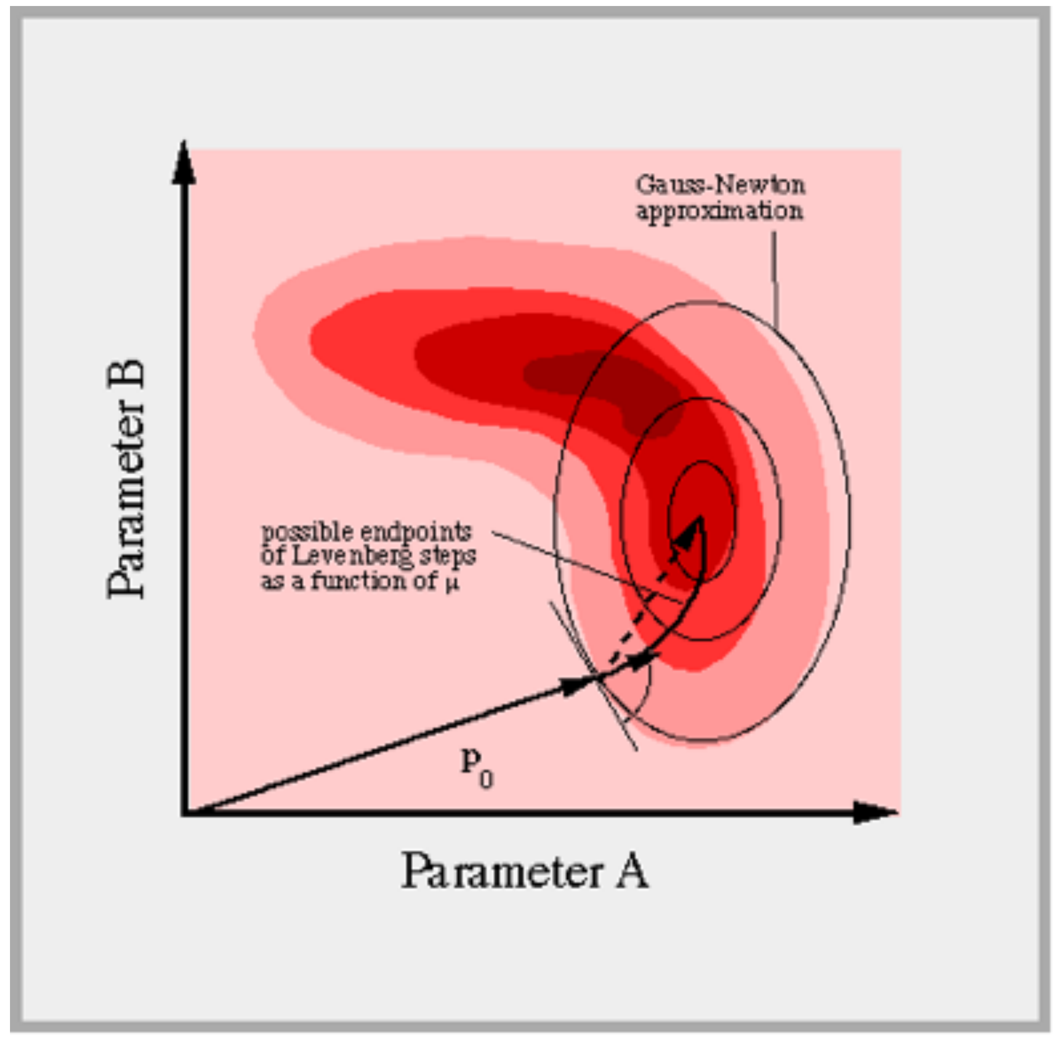
Function Minimization Algorithms

- Common interface class (`ROOT::Math::Minimizer`)
- Existing implementations available in ROOT as plug-ins:
  - **TMinuit** direct translation from Fortran code of MINUIT program
    - 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 algorithms from GSL and BFGS
  - **GSLMultiFit**: Levenberg-Marquardt (for 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
  - **RMinimiser**: based on optimisation algorithms from R (optim and optima packages)
- Easy to extend and add new implementations
- Possible to combine them (Minuit + Genetic)

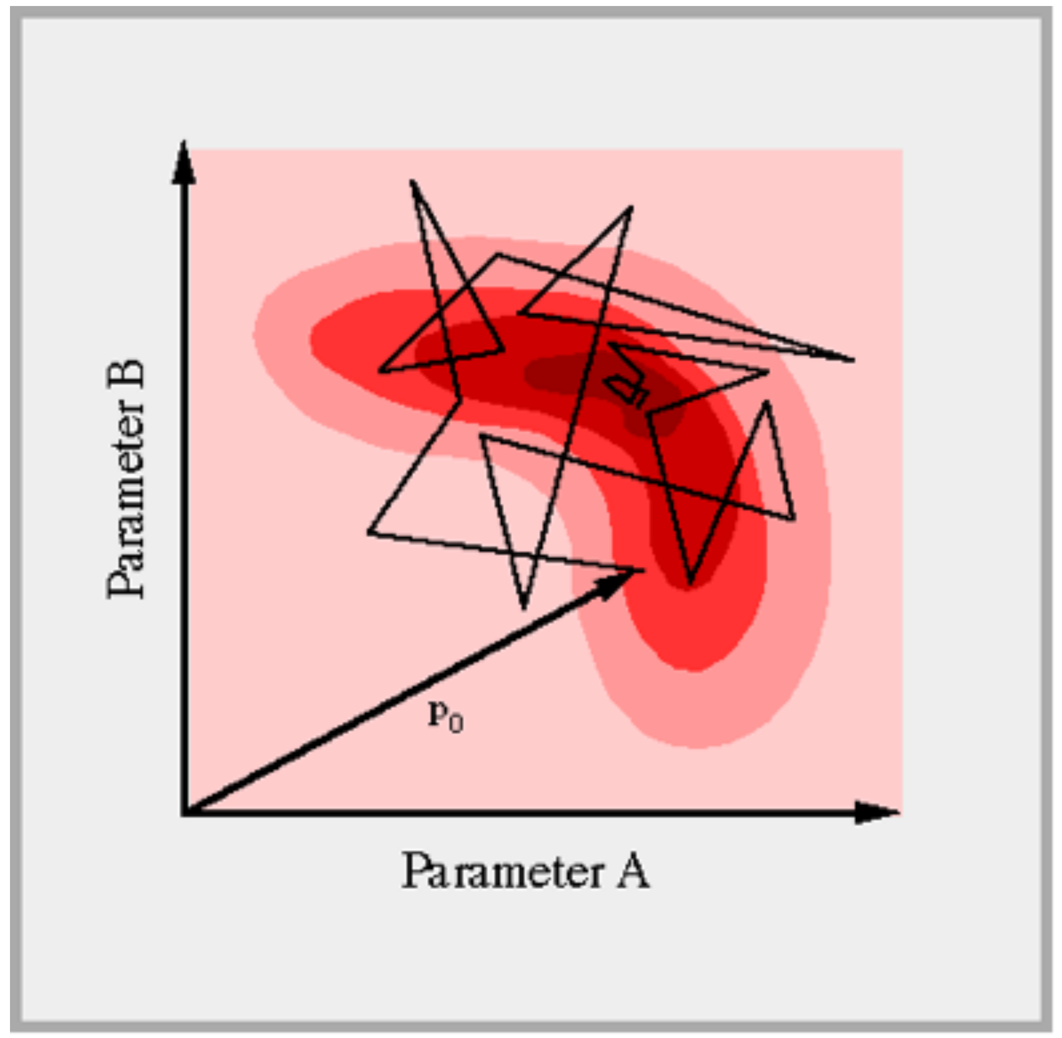
Minimization Techniques

- Methods like Minuit based on gradient can get stuck easily in local minima.
- Stochastic methods like simulated annealing or genetic algorithms can help to find the global minimum.

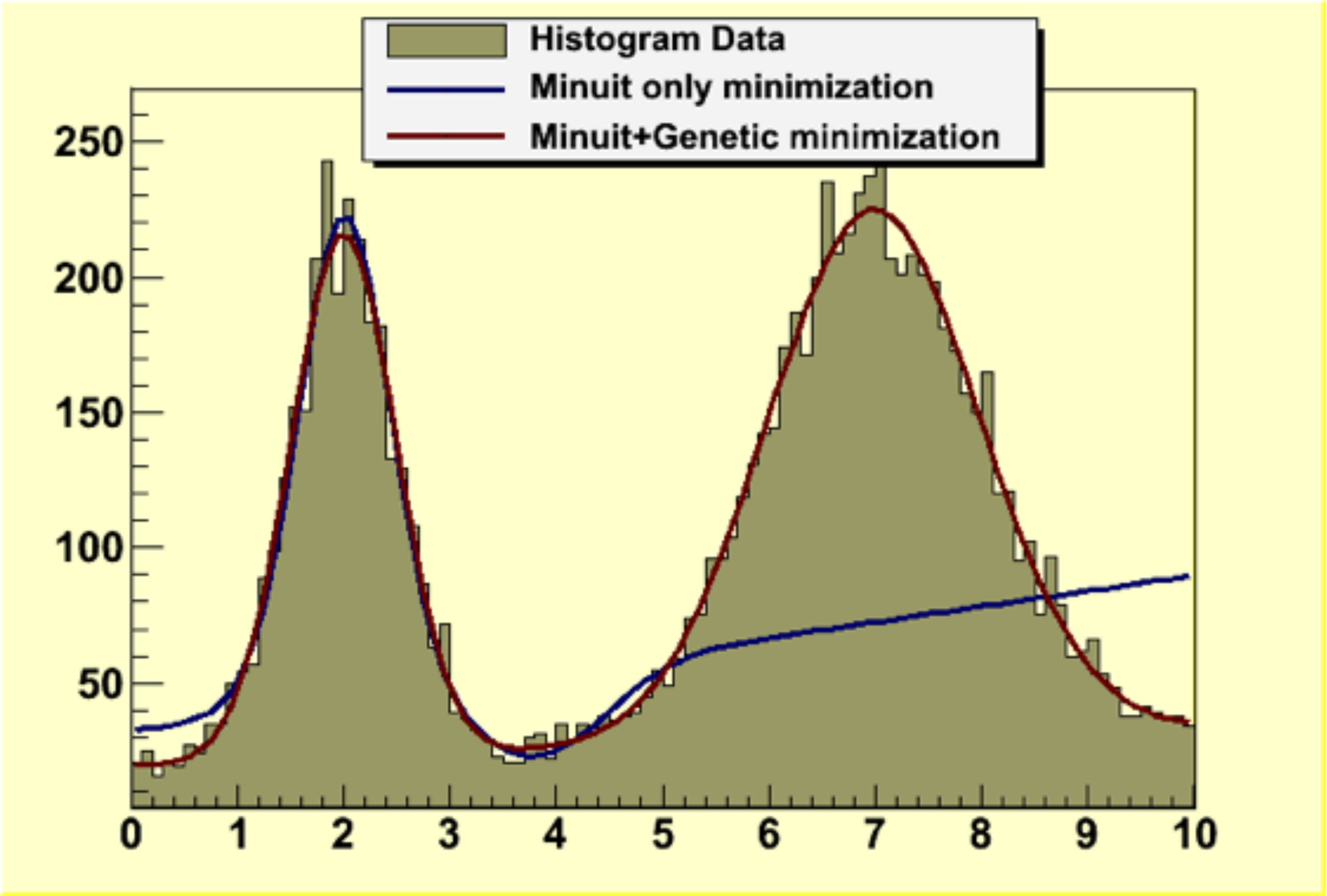
Quadratic Newton



Simulated Annealing



Example: Fitting 2 peaks in a spectrum



Comments on Minimization

- **Sometimes fits converge 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.

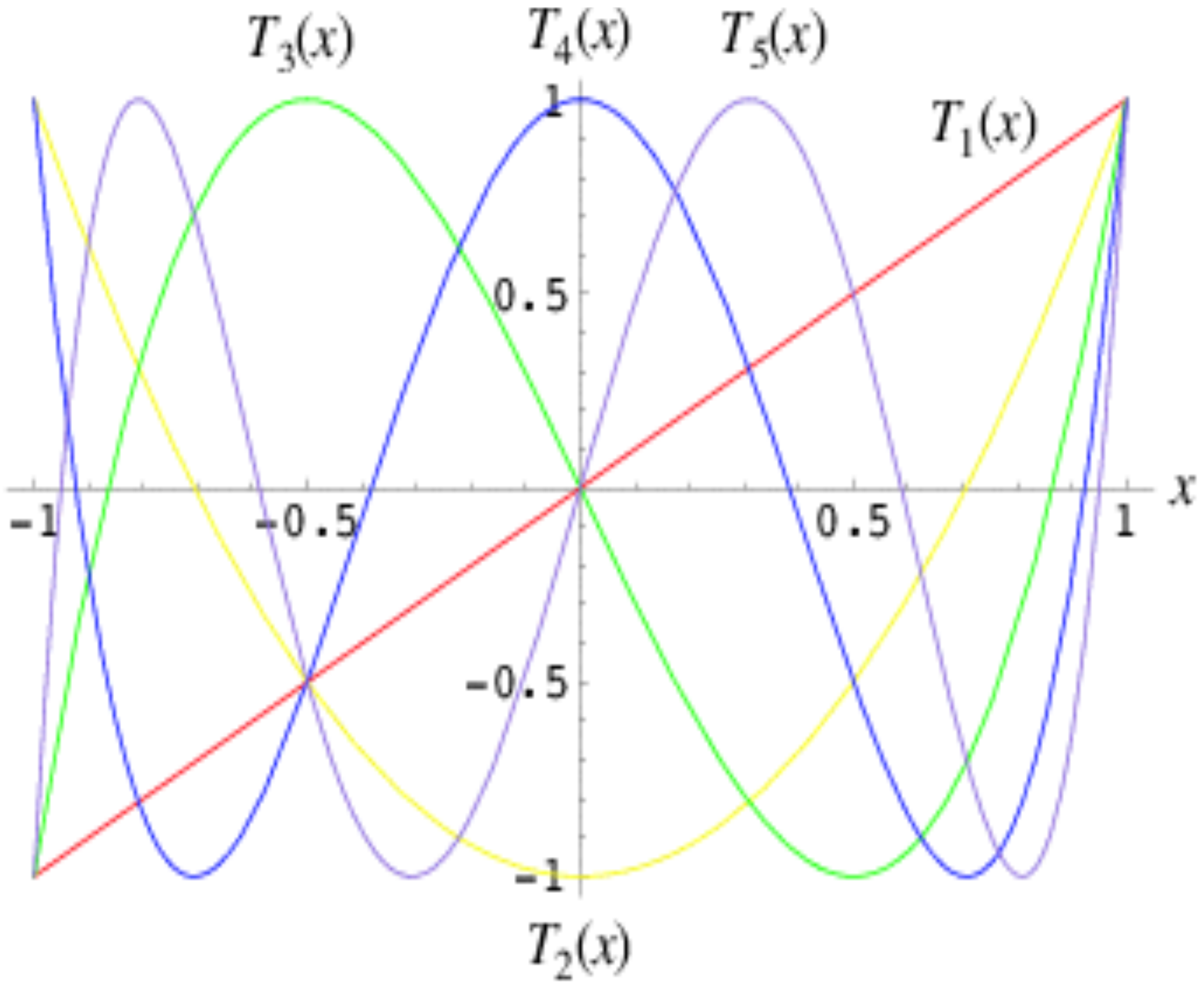
```
PARAMETER  CORRELATION COEFFICIENTS
      NO.   GLOBAL       1       2
       1   0.99835    1.000   0.998
       2   0.99835    0.998   1.000
```

Signs of trouble…

Mitigating fit stability problems

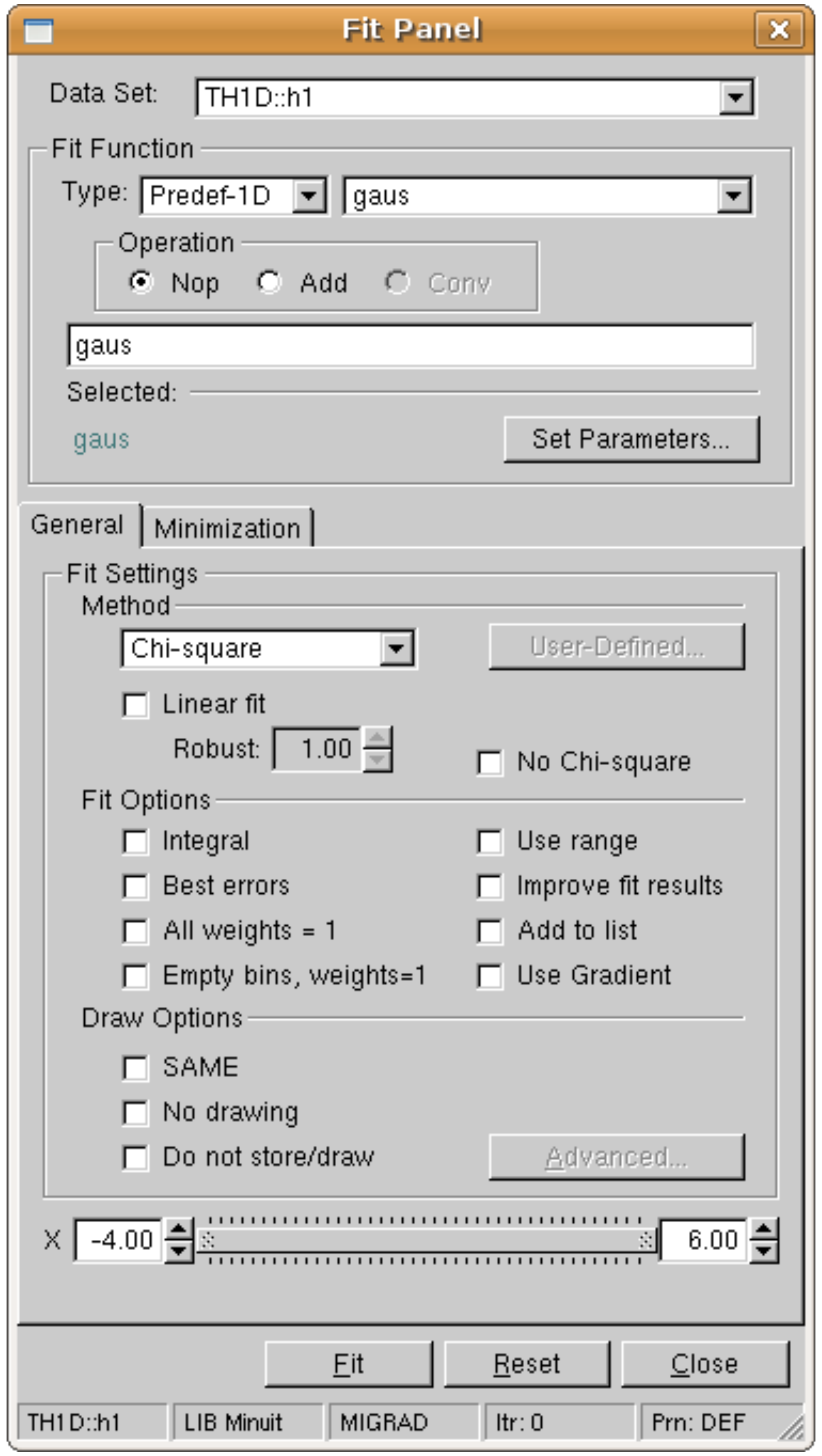
- When using a polynomial parametrization:
  - $a_0+a_1x+a_2x^2+a_3x^3$ nearly always results in strong correlations between the coefficients.
    - problems in fit stability and inability to find the right solution at high order
- This can be solved using a better polynomial parametrization:
  - e.g.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

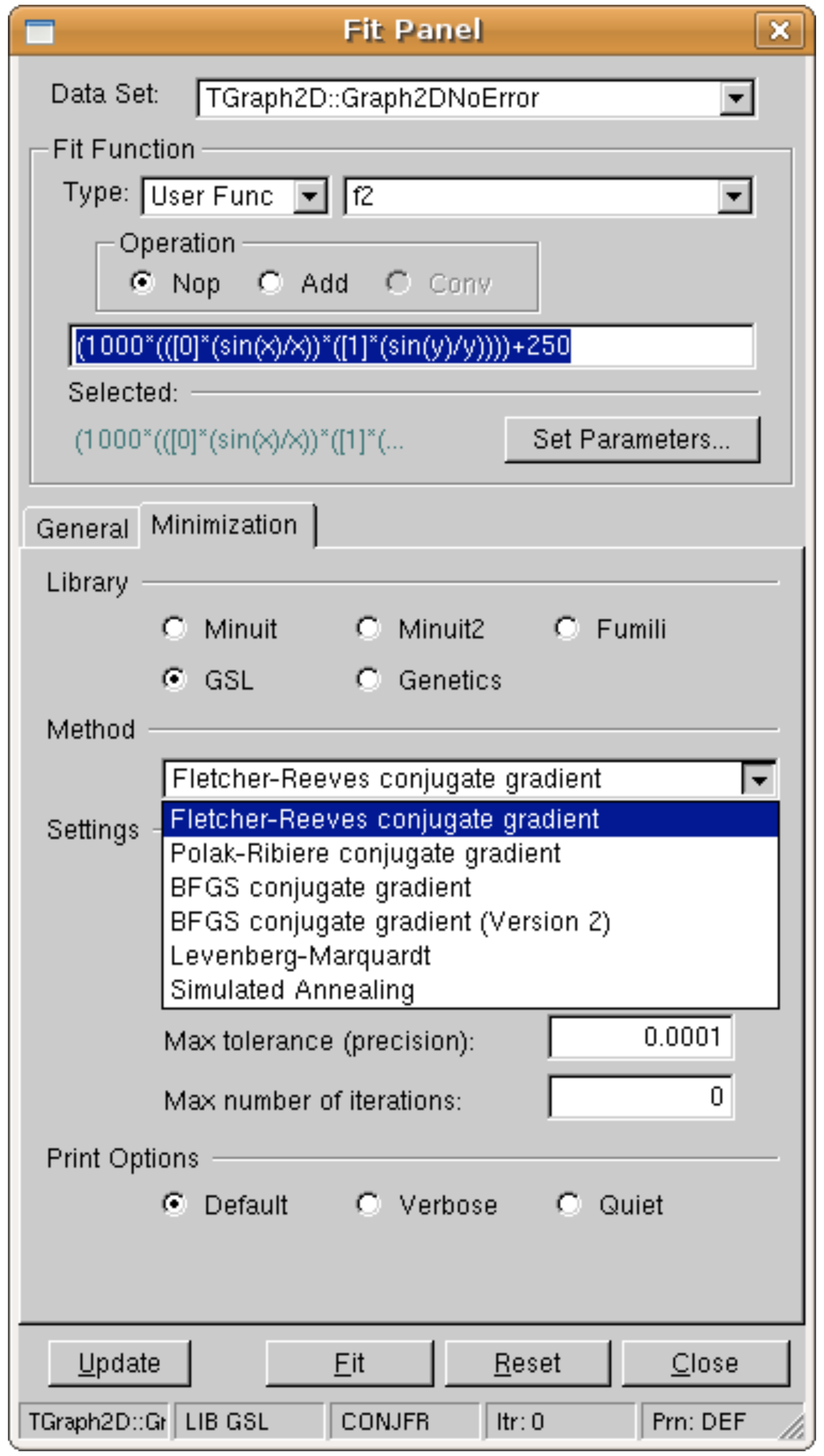
- The fitting in ROOT using the FitPanel GUI
  - GUI for fitting all ROOT data objects (histogram, graphs, trees)
- Using the GUI we can:
  - select data object to fit
  - choose (or create) fit model function
  - set initial parameters
  - choose:
    - fit method (likelihood, chi2)
    - fit options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



Fit Panel (2)

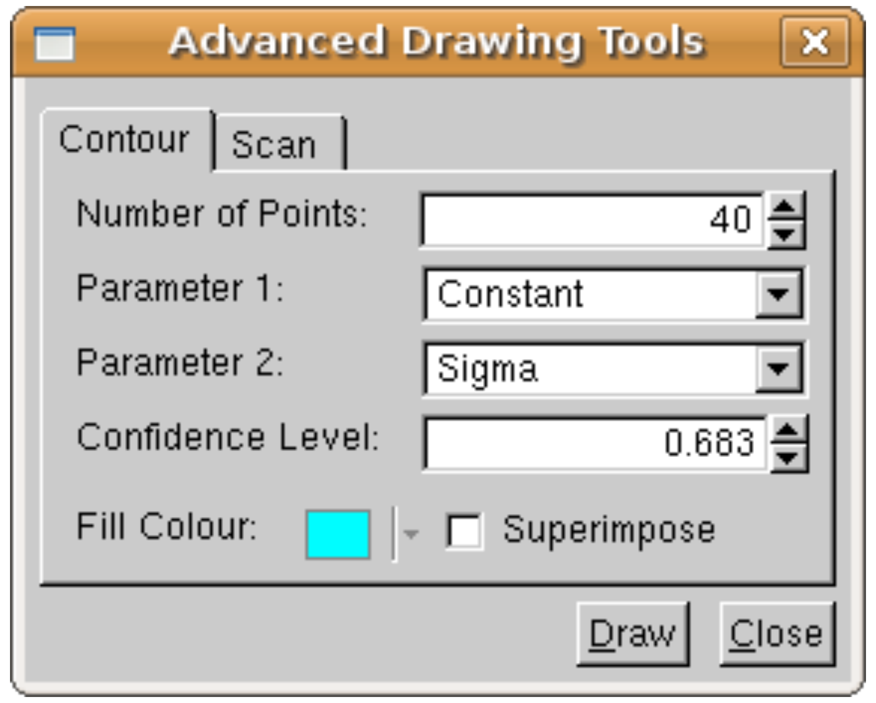
- The Fit Panel provides also extra functionality:

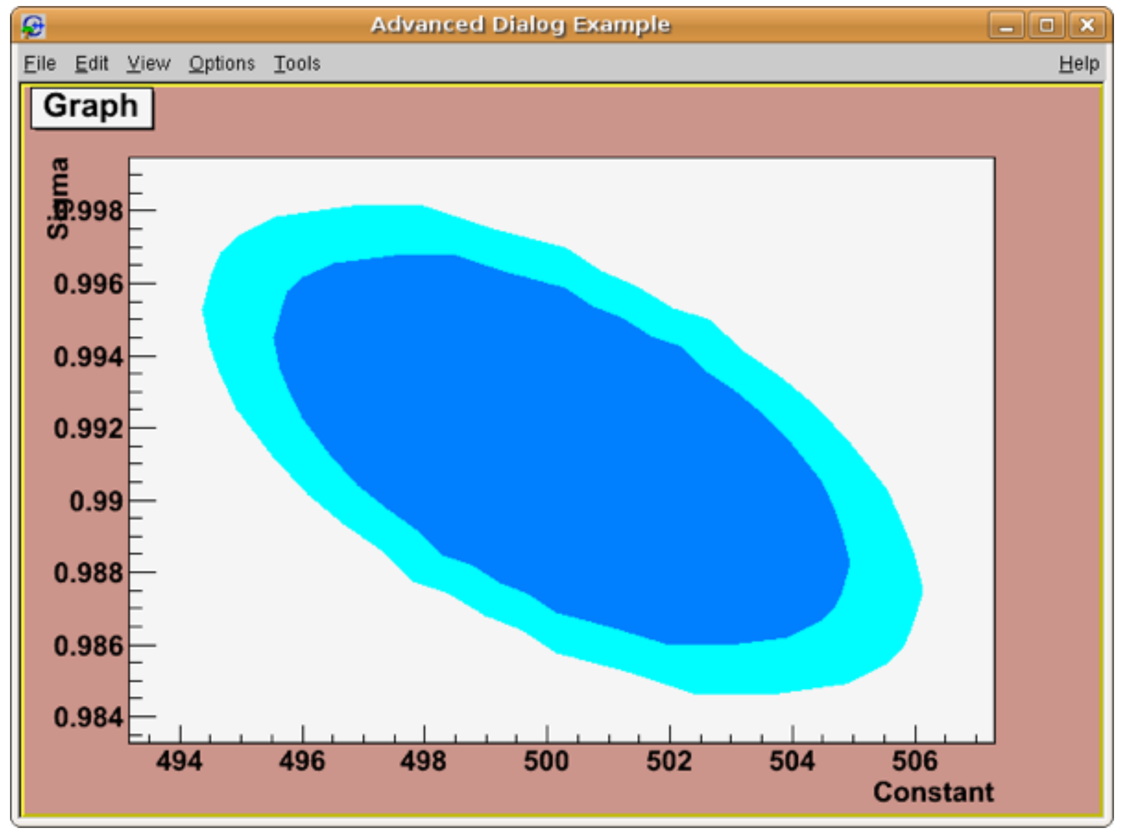
Control the minimization



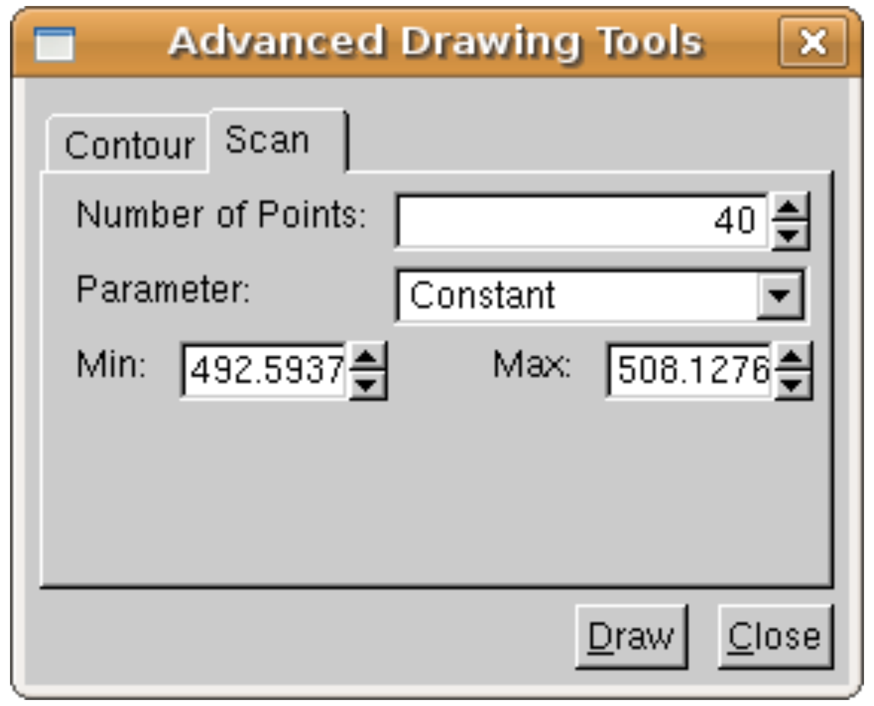
Advanced drawing tools

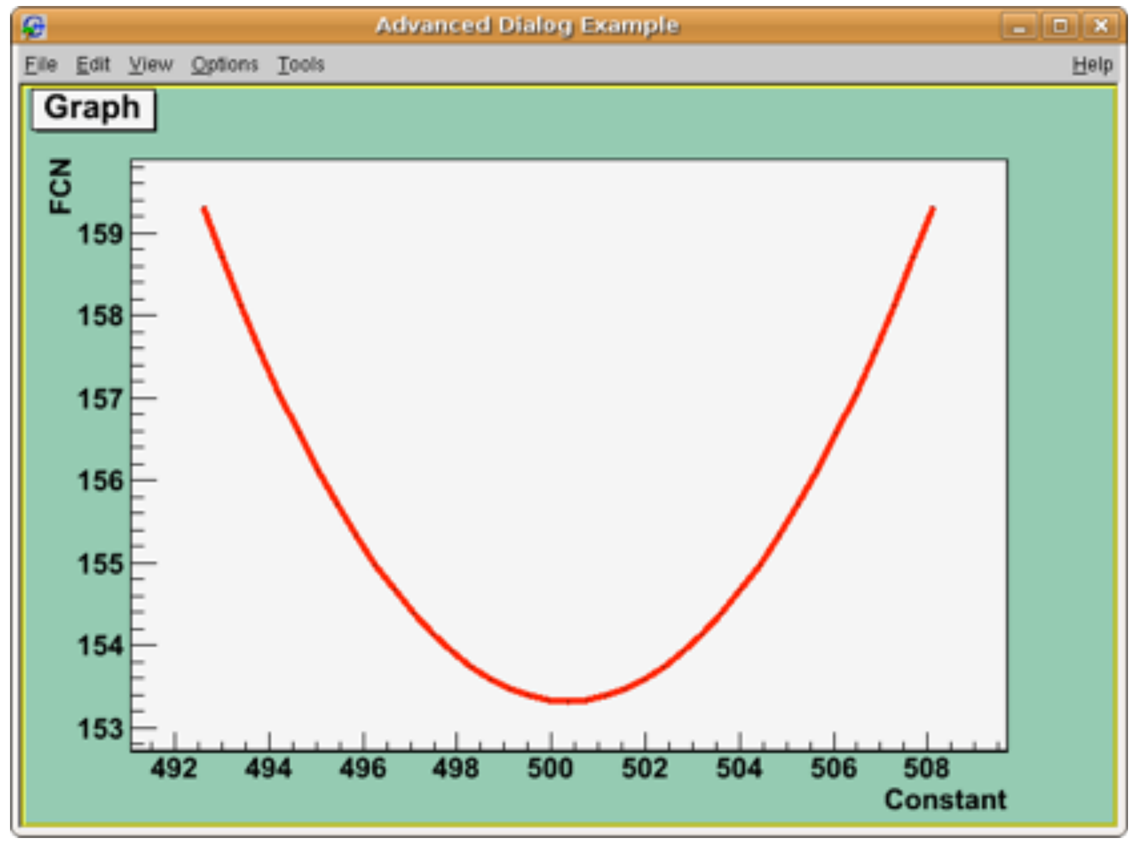
Contour plot



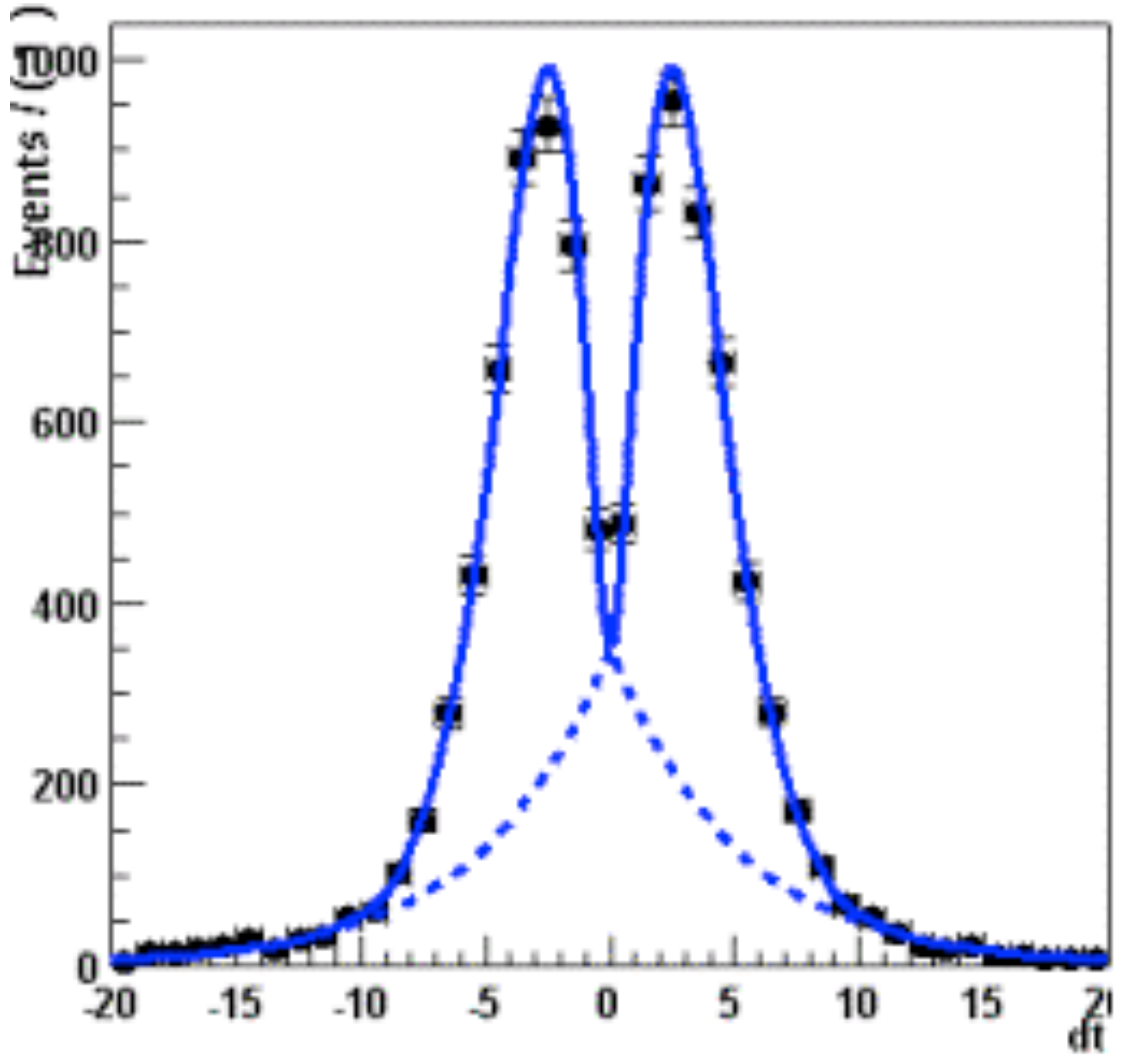


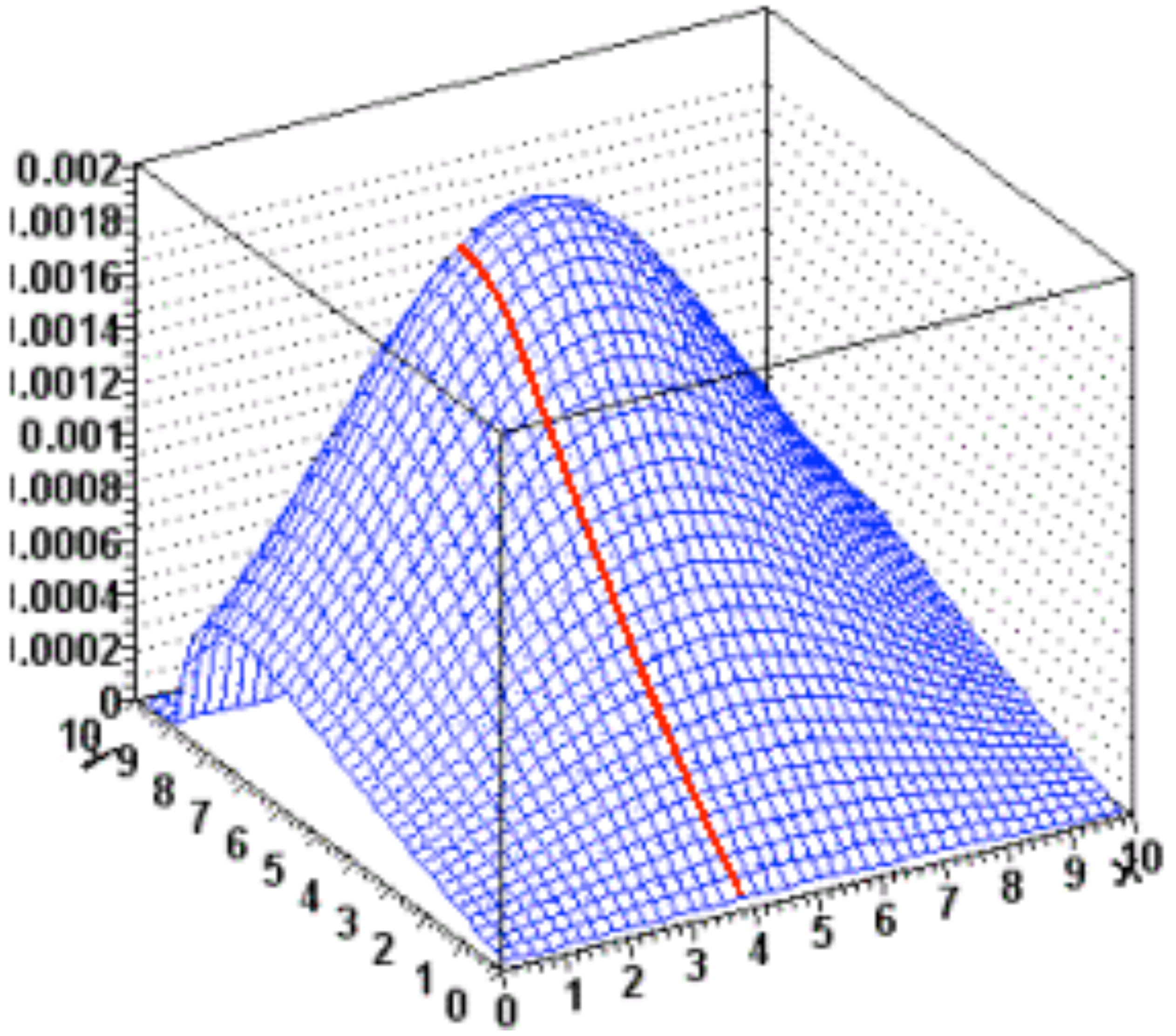
Scan plot of minimization function

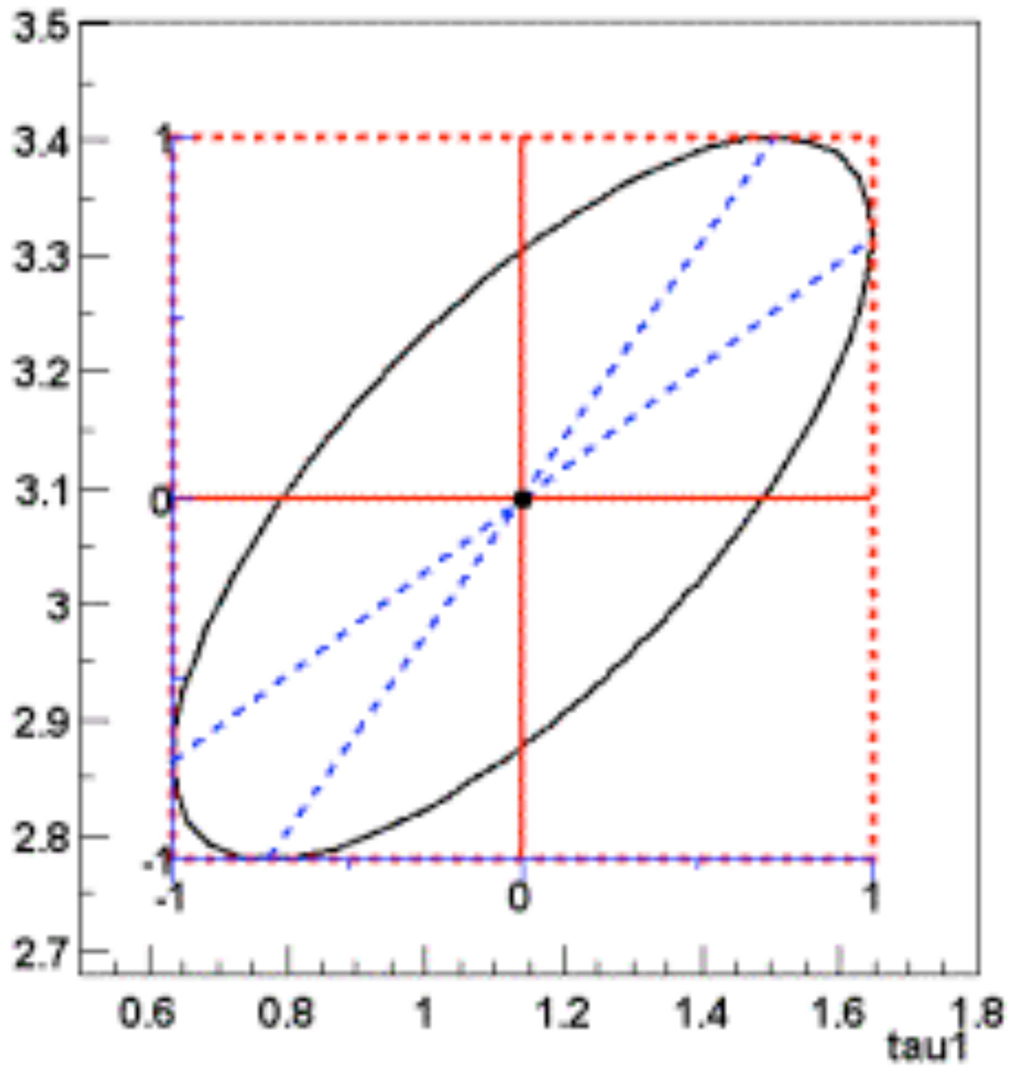




RooFit







Dr Lorenzo Moneta
CERN PH-SFT
CH-1211 Geneva 23
sftweb.cern.ch
root.cern.ch

Outline

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

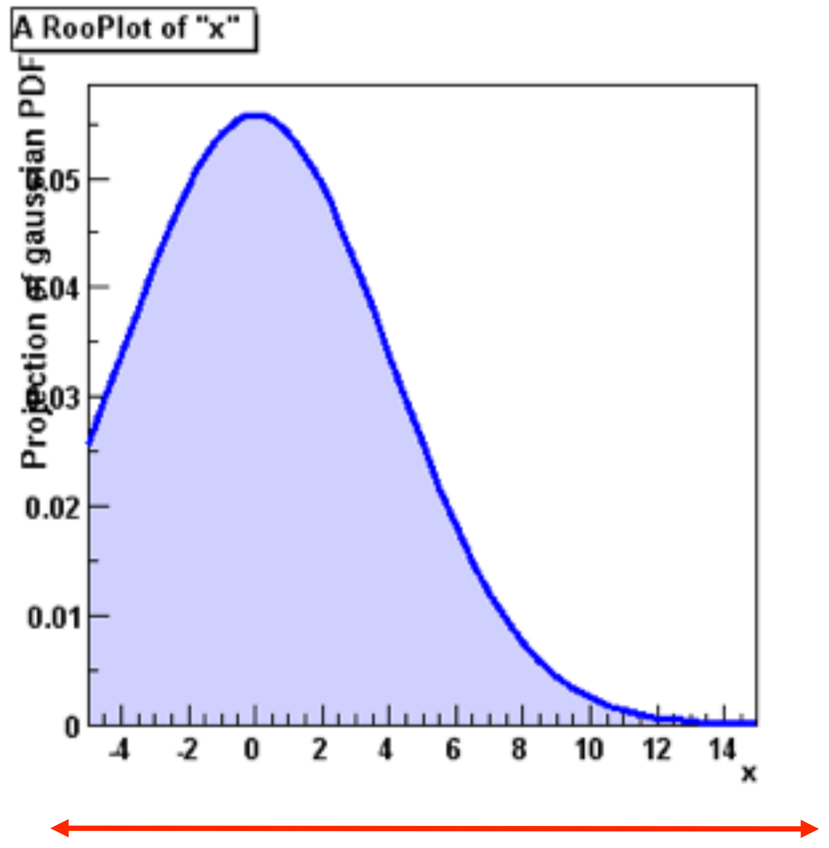
Material based on slides from W. Verkerke (author of RooFit)

- Exercises on RooFit:
  - building and fitting model

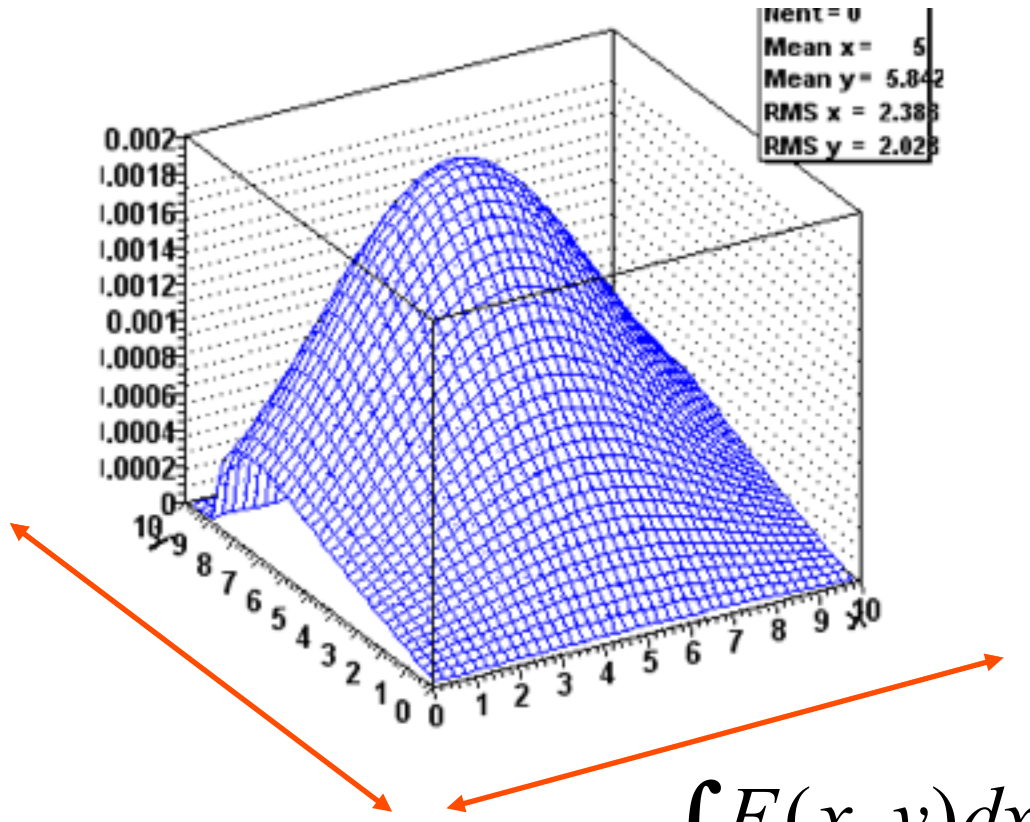
What is RooFit ?

- A toolkit distributed with ROOT and based on its core functionality.
- It is used to model distributions, which can be used for fitting and statistical data analysis.
  - model distribution of observable ***x*** in terms of parameters ***p***
    - probability density function (p.d.f.): $\mathcal{P}$`(x;p)`
    - p.d.f. are normalized over allowed range of observables ***x*** with respect to the parameters ***p***

$$\int_{\Omega} P(\vec{x};\vec{p})d\vec{x} = 1$$



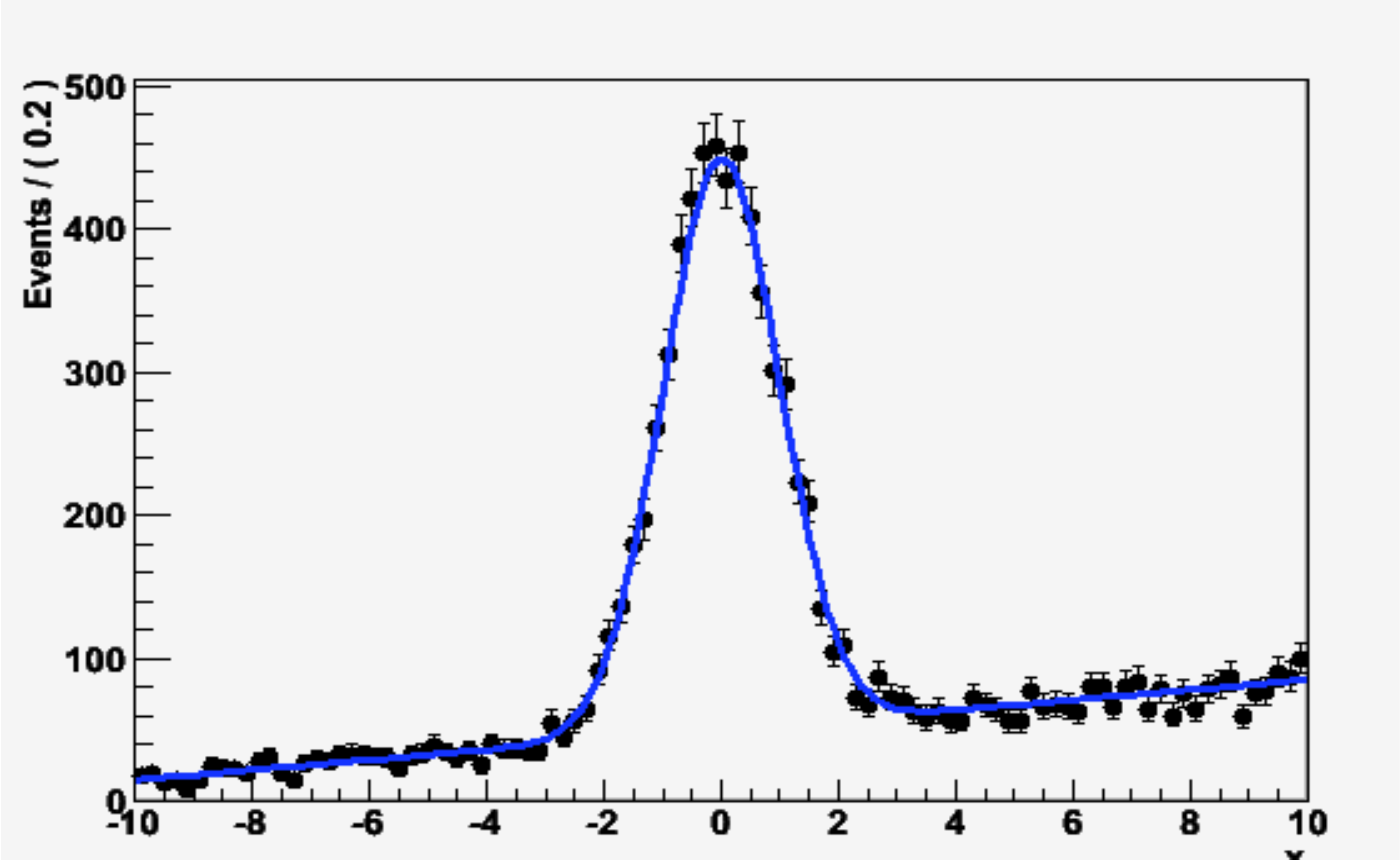
$$\int F(x)dx \equiv 1$$



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

Coding Probability Density Function

- How do we formulate the p.d.f. in ROOT
  - For 'simple' problems (gauss, polynomial) this is easy



  - But if we want to do complex likelihood fits using non-trivial functions and composing several p.d.f., or to work with multidimensional functions it becomes difficult to do it in ROOT

Why RooFit ?

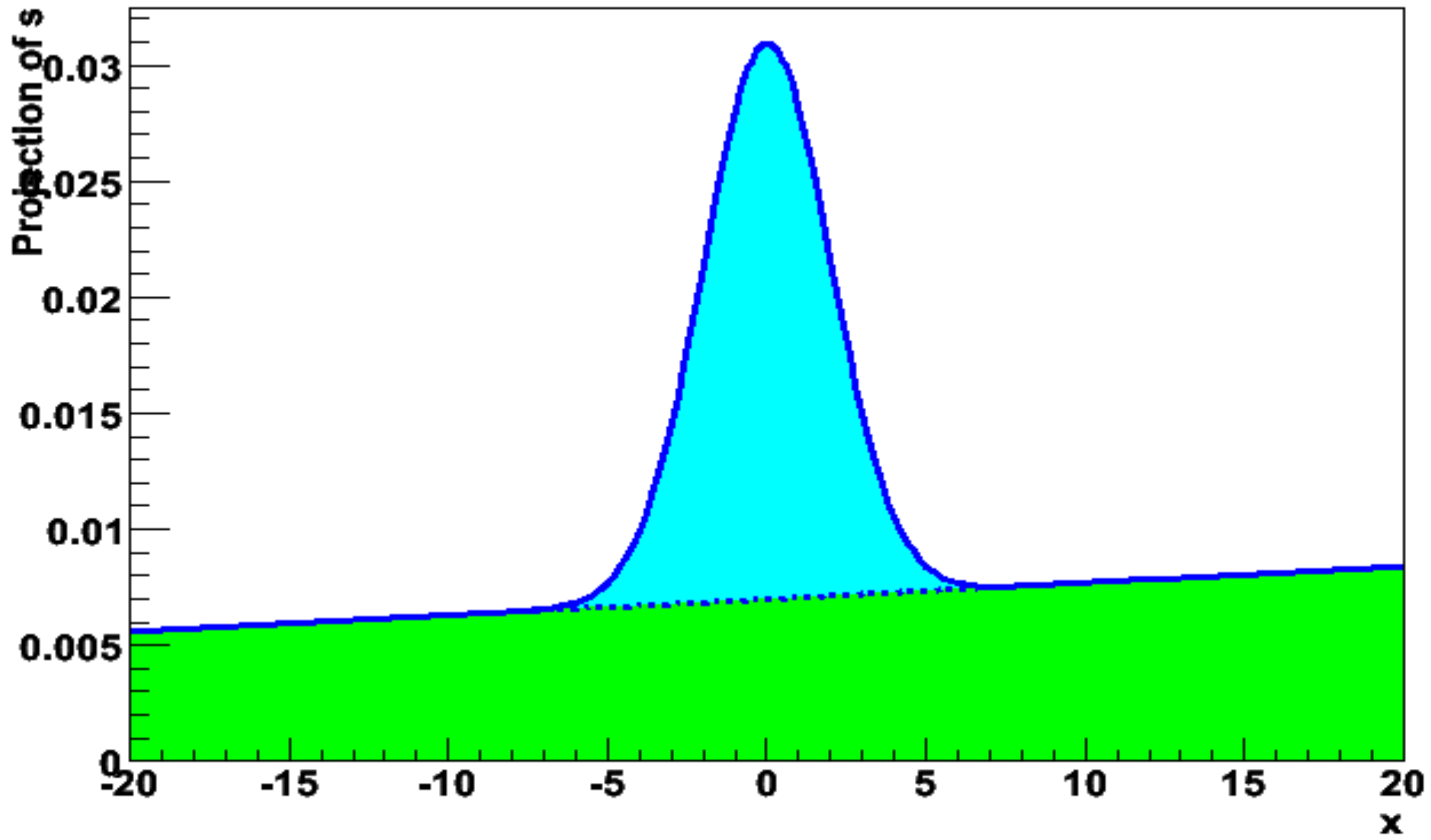
- ROOT can handle complicated functions but it might require writing large amount of code
- Normalization of p.d.f. not always trivial
  - RooFit does it automatically
- In complex fit, computation performance important
  - need to optimize code for acceptable performance
  - built-in optimization available in RooFit
    - evaluation of model parts 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

Math – Functions vs probability density functions

- Why use *probability density* functions rather than 'plain' functions to model the data?
  - *Easier to interpret the models*.
    If Blue and Green pdf are each guaranteed to be normalized to 1, then fractions of Blue,Green can be cleanly interpreted as #events
  - Many statistical techniques only function properly with p.d.f.
    (e.g maximum likelihood fits)



- What is difficult with p.d.f ?
  - The normalization can be hard to calculate
    (e.g. it can be different for each set of parameter values p)
    - In >1 dimension (numeric) integration can be particularly hard
  - RooFit aims to simplify these tasks

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 *Gaus(x,m,s)*

```
RooGaussian g

RooRealVar x     RooRealVar s

RooRealVar m
```

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)
```

The simplest possible example

- We make a Gaussian p.d.f. with three variables: mass, mean and sigma

Name of object

Title of object

Initial range

Objects representing a ‘real’ value.

```
RooRealVar x(“x”,”Observable”,-10,10) ;
RooRealVar mean(“mean”,”B0 mass”,0.00027);
RooRealVar sigma(“sigma”,”B0 mass width”,5.2794) ;
```

Initial value

PDF object

```
RooGaussian model(“model”,”signal pdf”,x,mean,sigma)
```

References to variables

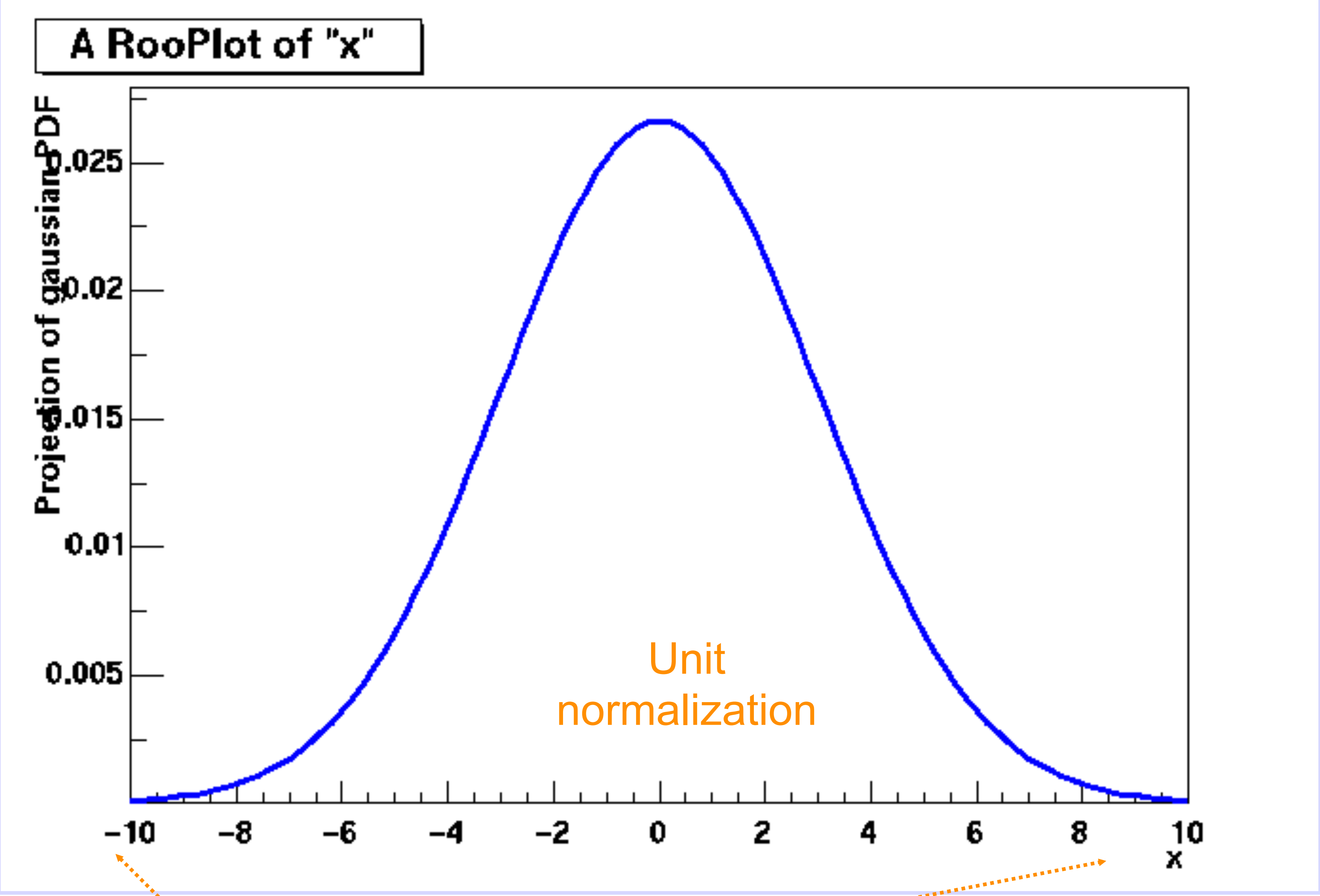
Creating and plotting a Gaussian p.d.f

Setup gaussian PDF and plot

```
// Create an empty plot frame
RooPlot* xframe = x.frame() ;

// Plot model on frame
model.plotOn(xframe) ;

// Draw frame on canvas
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`

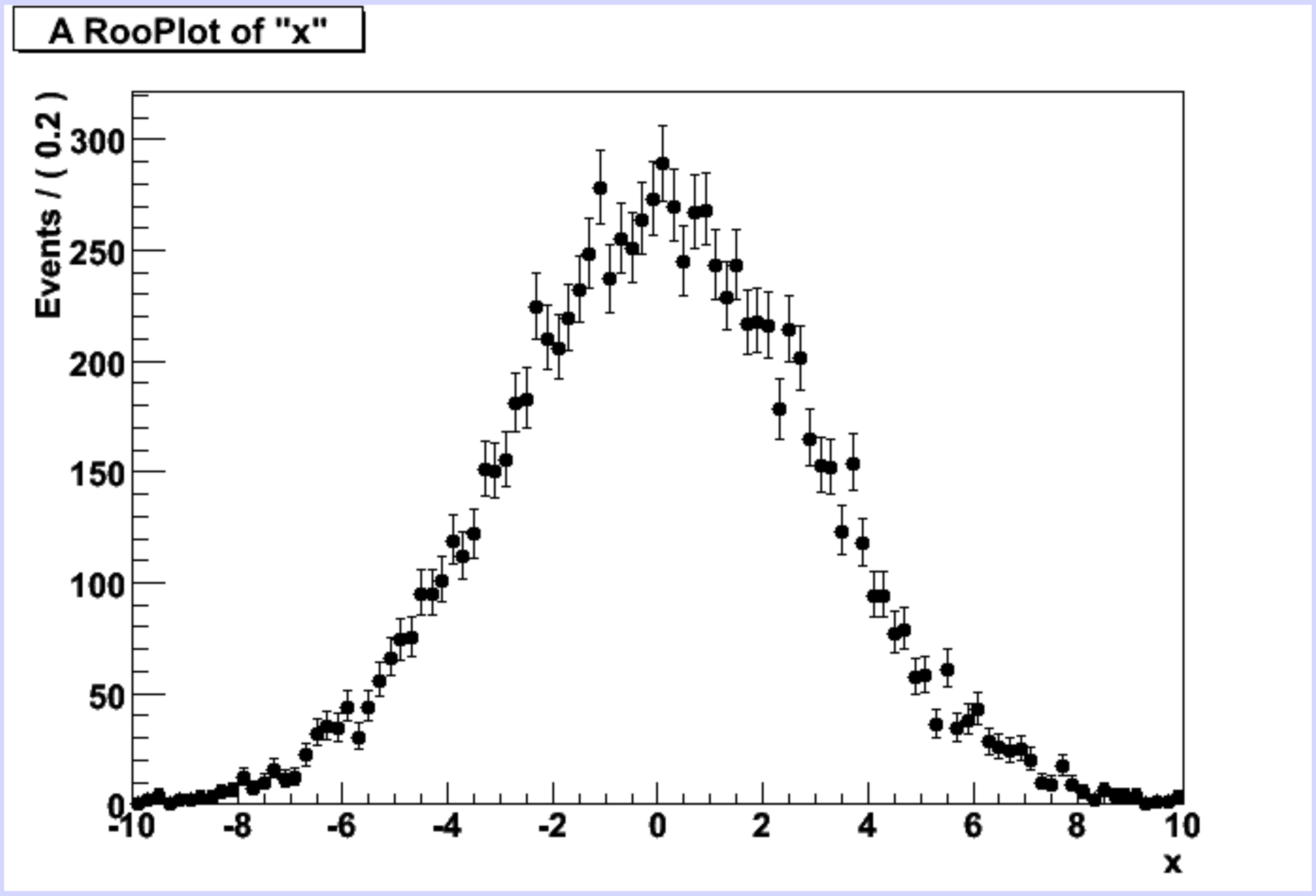
Basics – Generating toy MC events

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

```
// Generate an unbinned toy MC set
RooDataSet* data = gauss.generate(x,10000) ;

// Generate an binned toy MC set
RooDataHist* data = gauss.generateBinned(x,10000) ;
```

Can generate both binned and unbinned datasets



Data visualization

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

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

  – Imports values, binning definition *and* errors (if defined)

  – Specify a **RooArgList** of observables when importing a TH2/3.

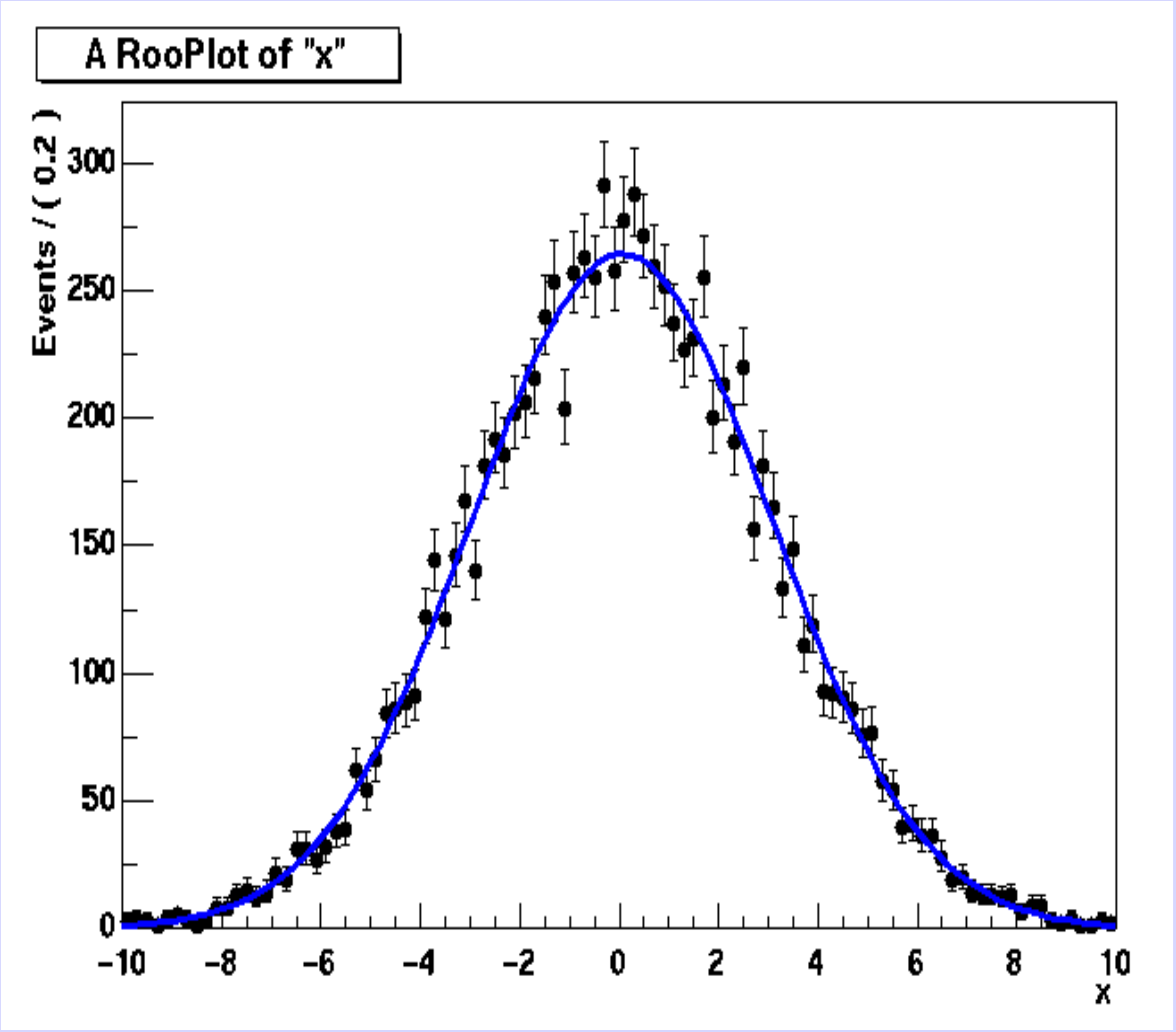
Basics: Fitting the data

- 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();
```



PDF automatically normalized to dataset

Exercises working with RooFit

- Create a Gaussian p.d.f, generate some toy data and fit it
- Extra:
  - Play with some other p.d.f
    - e.g. Exponential pdf
    - or some other p.d.f you want.
    - You can find several pdf in roofit reference documentations
      - http://root.cern.ch/root/html/ROOFIT_ROOFIT_Index.html
      - (all class names in RooFit starts with “Roo”)

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)
```

Provides a factory to auto-generate objects from a math-like language

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

We will work in the examples 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 functions into a workspace
  - When importing a function, 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

RooPlot* frame = w.var("x")->frame() ;

w.pdf("f")->plotOn(frame) ;
```

Using the workspace

- Workspace can be written to a file with all its contents
  - Writing workspace and contents to file

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

- Organizing your code – Separate construction and use of models

```
void driver() {
  RooWorkspace w("w") ;
  makeModel(w) ;
  useModel(w) ;
}

void makeModel(RooWorkspace& w) {
  // Construct model here
}

void useModel(RooWorkspace& w) {
  // Make fit, plots etc here
}
```

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)
// equivalent to RooGaussian(“g”,”g”,x,mean,sigma)

Polynomial::p(x,{a0,a1})
// equivalent to  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

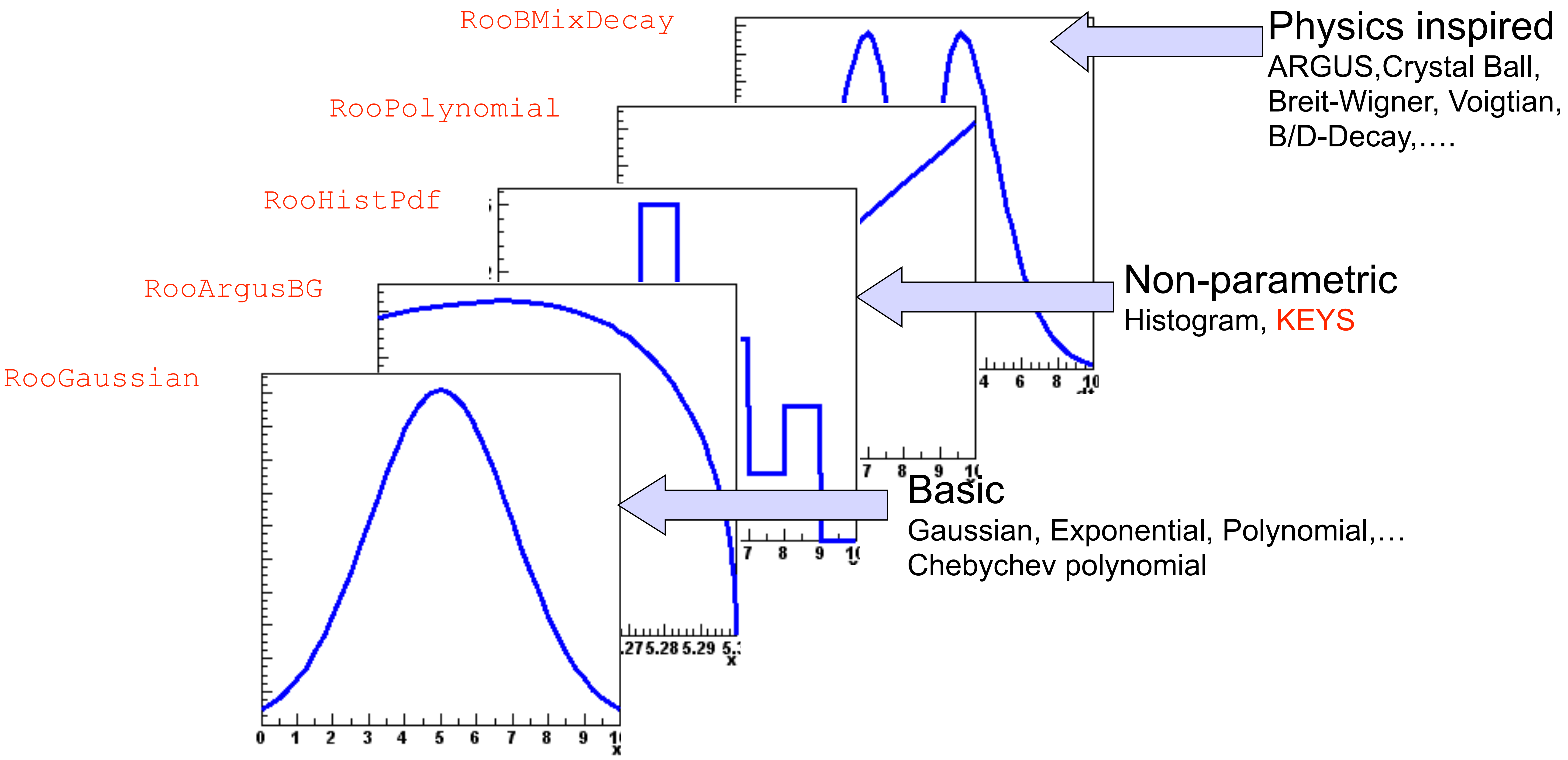
```
Gaussian::g(x[-10,10],0,3)
```

  - It is not required to give component objects a name, e.g.

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

Model building

- RooFit provides a collection of compiled standard PDF classes



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

(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	`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)

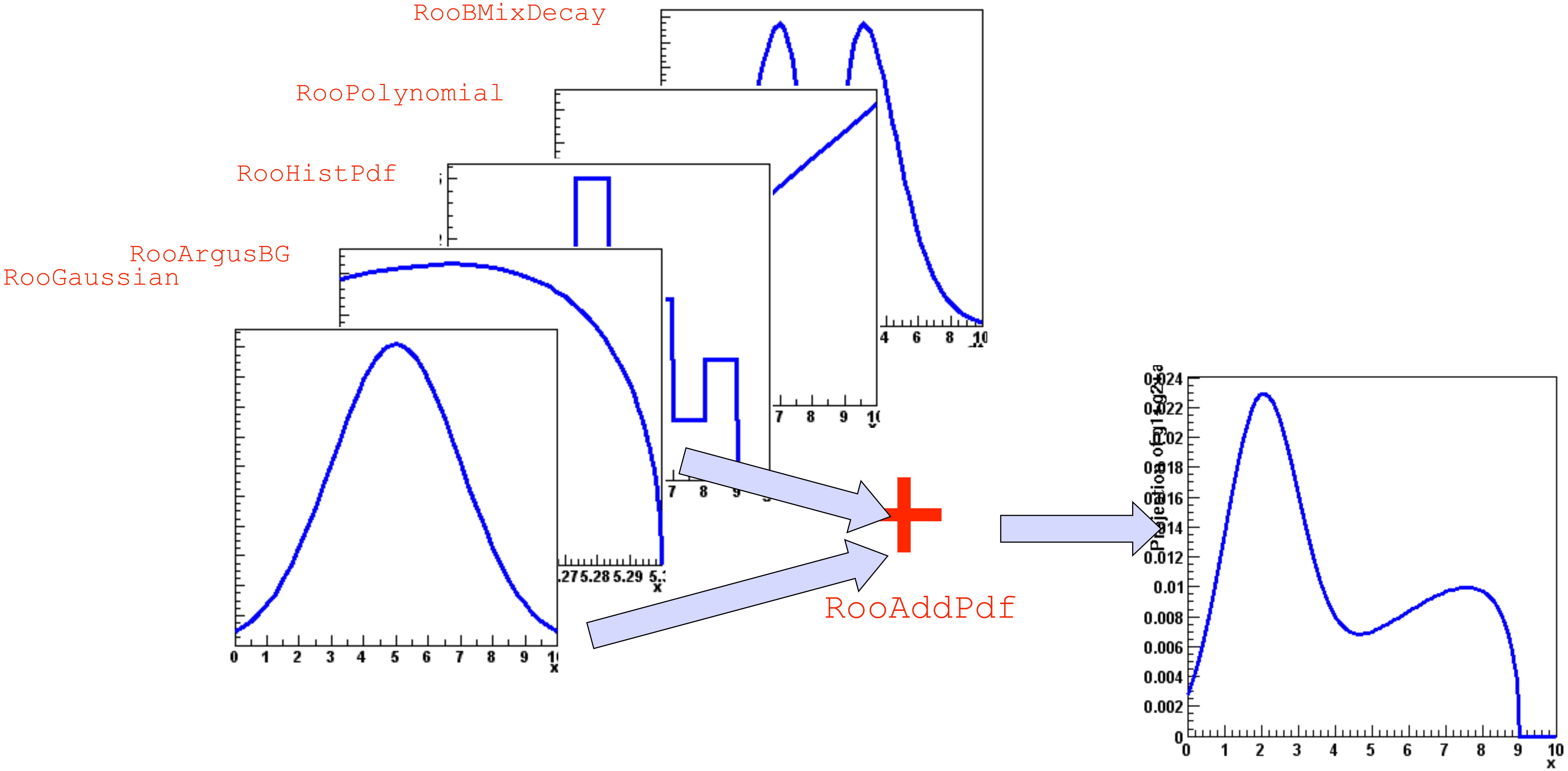
```
w.factory("expr::w('(1-D)/2',D[0,1])") ;
```

  – 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`**



Adding p.d.f.s – Factory syntax

- Additions created through a SUM expression

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

$$S(x) = fF(x) + (1-f)G(x)$$

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

  – Note that last PDF does not have an associated fraction in case of floating overall normalization
    - when the normalization is fitted from the observed events

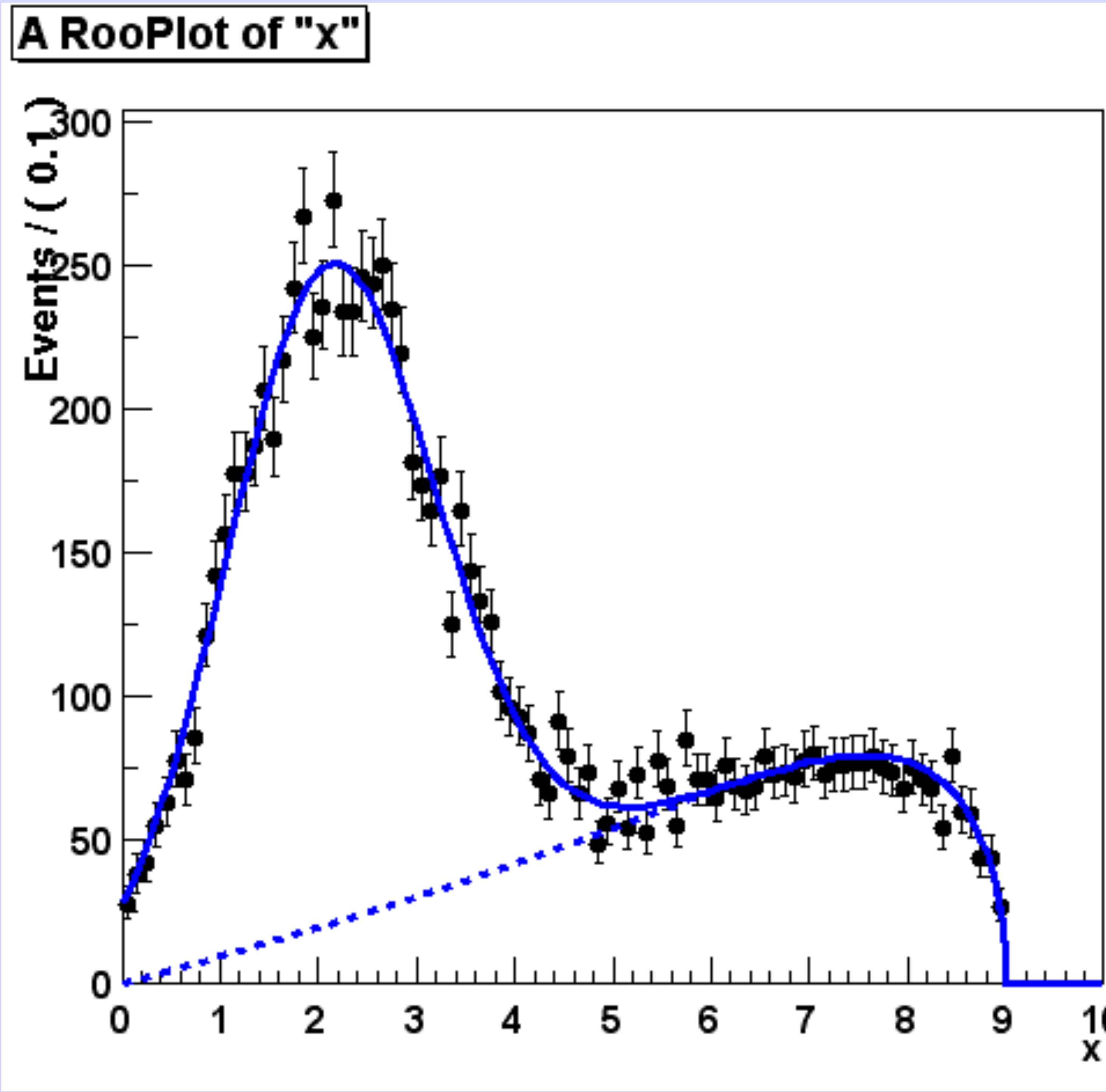
- Complete example

```
w.factory(“Gaussian::gauss1(x[0,10],mean1[2],sigma[1]”) ;
w.factory(“Gaussian::gauss2(x,mean2[3],sigma)”) ;
w.factory(“ArgusBG::argus(x,k[-1],9.0)”) ;

w.factory(“SUM::sum(g1frac[0.5]*gauss1, g2frac[0.1]*gauss2, argus)”)
```

Plotting Components of a p.d.f.

- 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 p.d.f.s
  - Component plotting




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

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

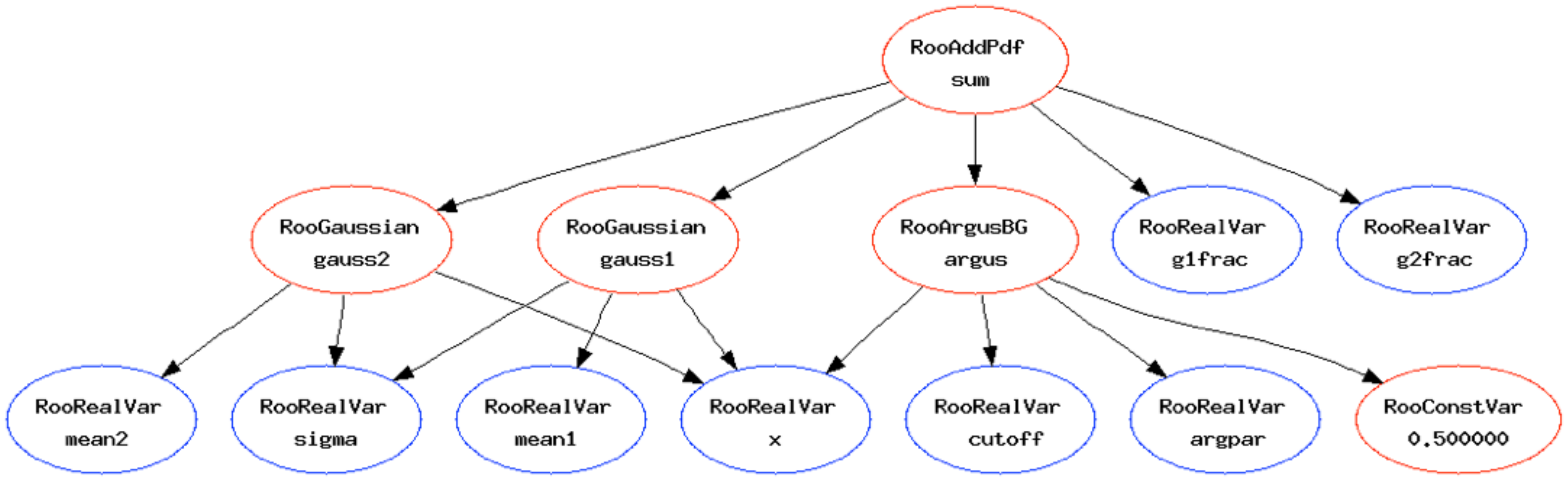
Operations on specific to composite pdfs

- Tree printing mode of workspace reveals component structure

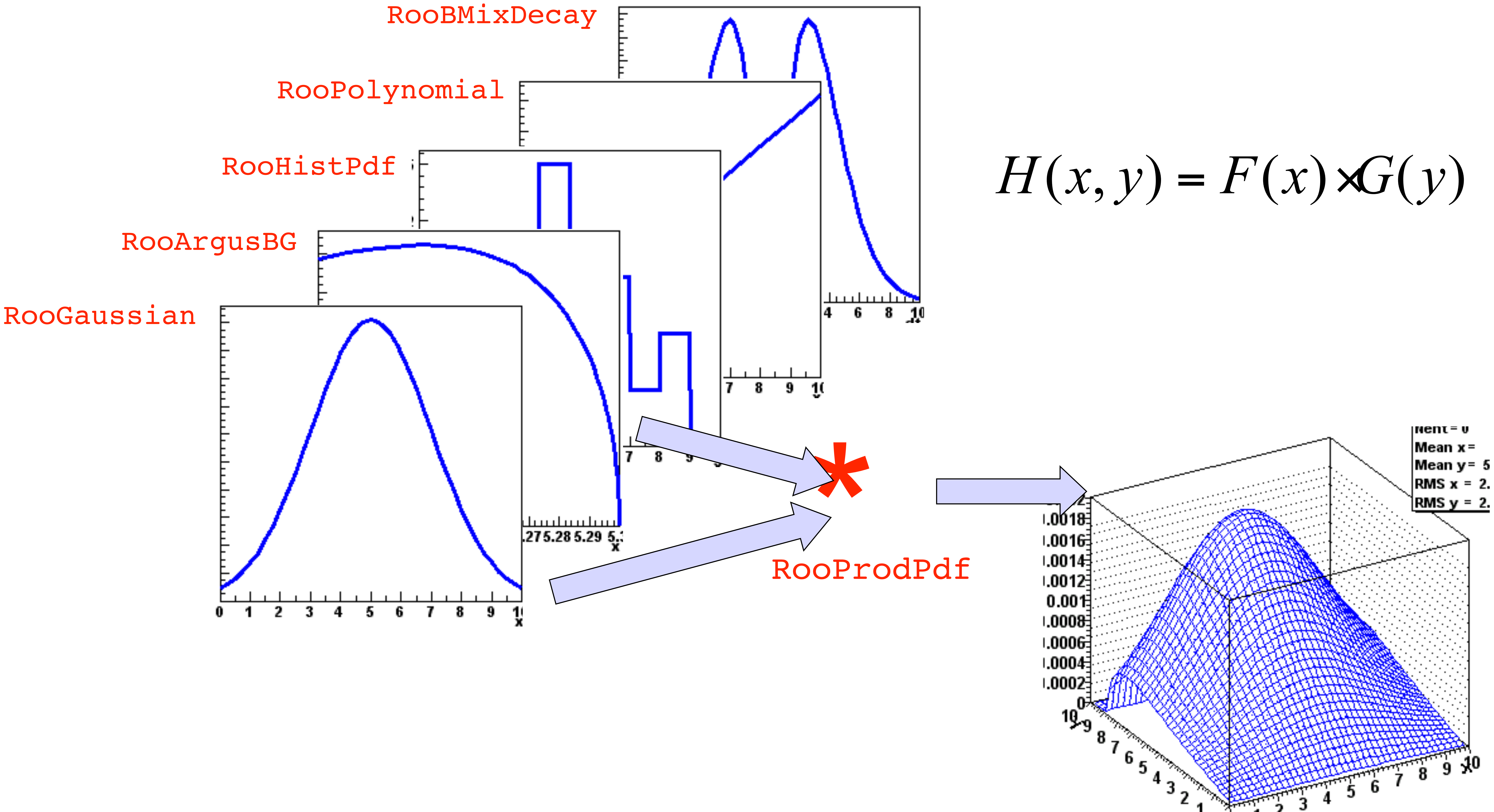
```
w.pdf("sum")->Print("t");
  RooAddPdf::sum[ g1frac * g1 + g2frac * g2 + [%] * argus ] = 0.0687785
      RooGaussian::g1[ x=x mean=mean1 sigma=sigma ] = 0.135335
      RooGaussian::g2[ x=x mean=mean2 sigma=sigma ] = 0.011109
      RooArgusBG::argus[ m=x m0=k c=9 p=0.5 ] = 0
```

  – Can also make input files for GraphViz visualization

```
w.pdf("sum")->graphVizTree("myfile.dot");
```



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 | **nD**

$$H(x,y) = F(x) \times G(y) \qquad H(x^{\{i\}}) = \prod_i F^{\{i\}}(x^{\{i\}})$$

  – 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)") ;
```

Composition of p.d.f.s

- 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

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")

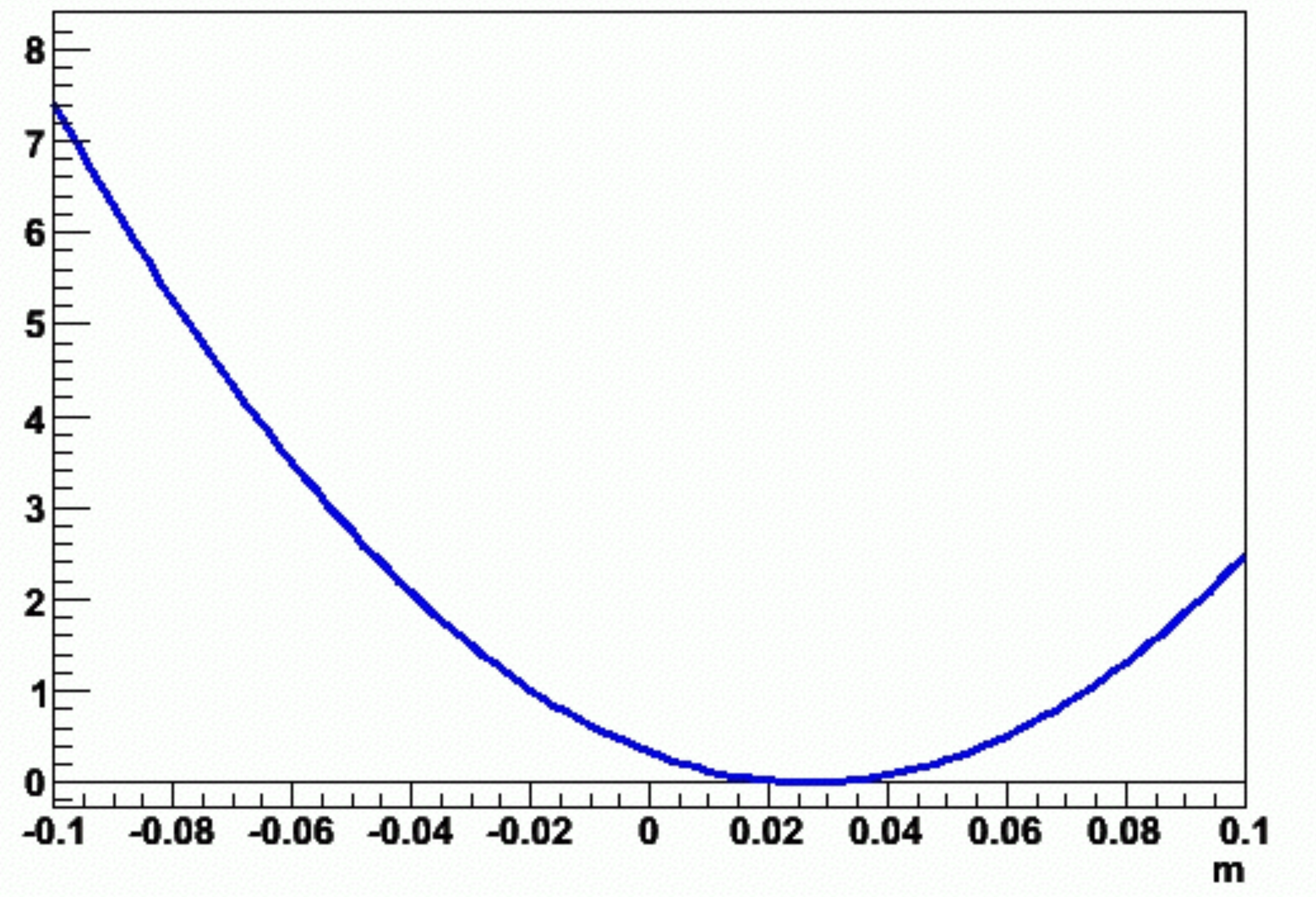
```
RooDataSet *dataA, *dataB ;
RooDataSet dataAB("dataAB","dataAB",
                  RooArgSet(*w.var("x"),*w.cat("index")),
                  Index(*w.cat("index")),
                  Import("A",*dataA),Import("B",*dataB)) ;
```

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)
```

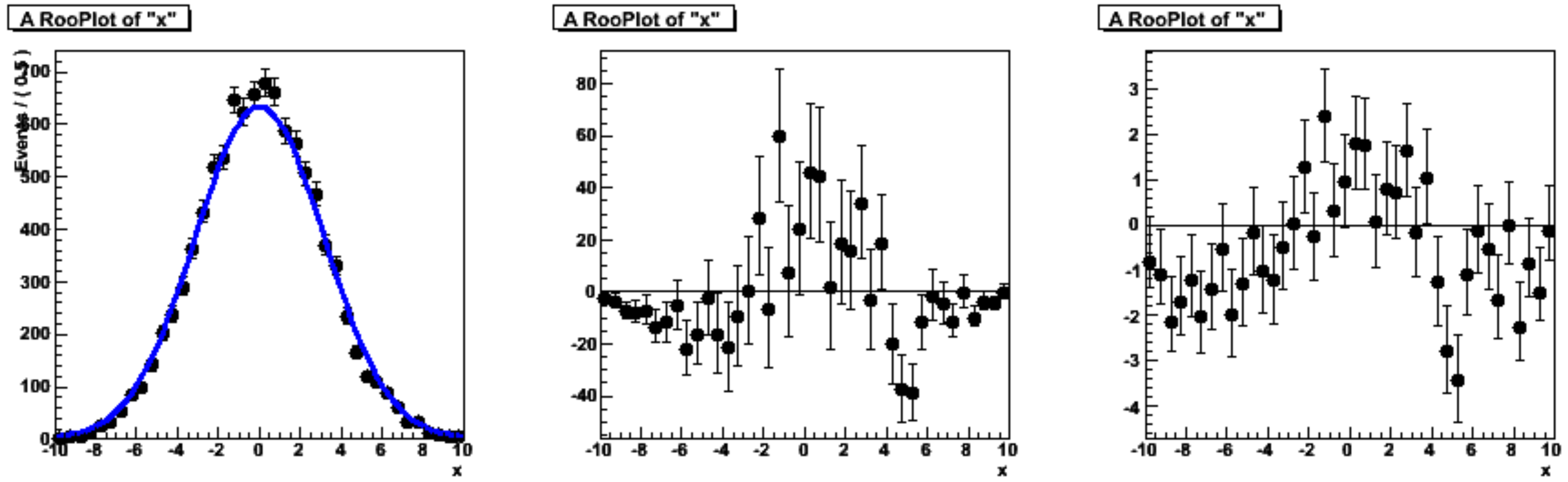
  - Also other minimizers (Minuit, GSL etc) supported
  - N.B. Different minimizer can also be used from RooAbsPdf::fitTo

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

How do you know if your fit was 'good'

- Goodness-of-fit broad issue in statistics (we will see maybe later)
- For one-dimensional fits, a χ^2 is usually the right thing to do
  - Some tools implemented in `RooPlot` to be able to calculate χ^2/`ndf` of curve w.r.t data

```
double chi2 = frame->chisquare(nFloatParam);
```

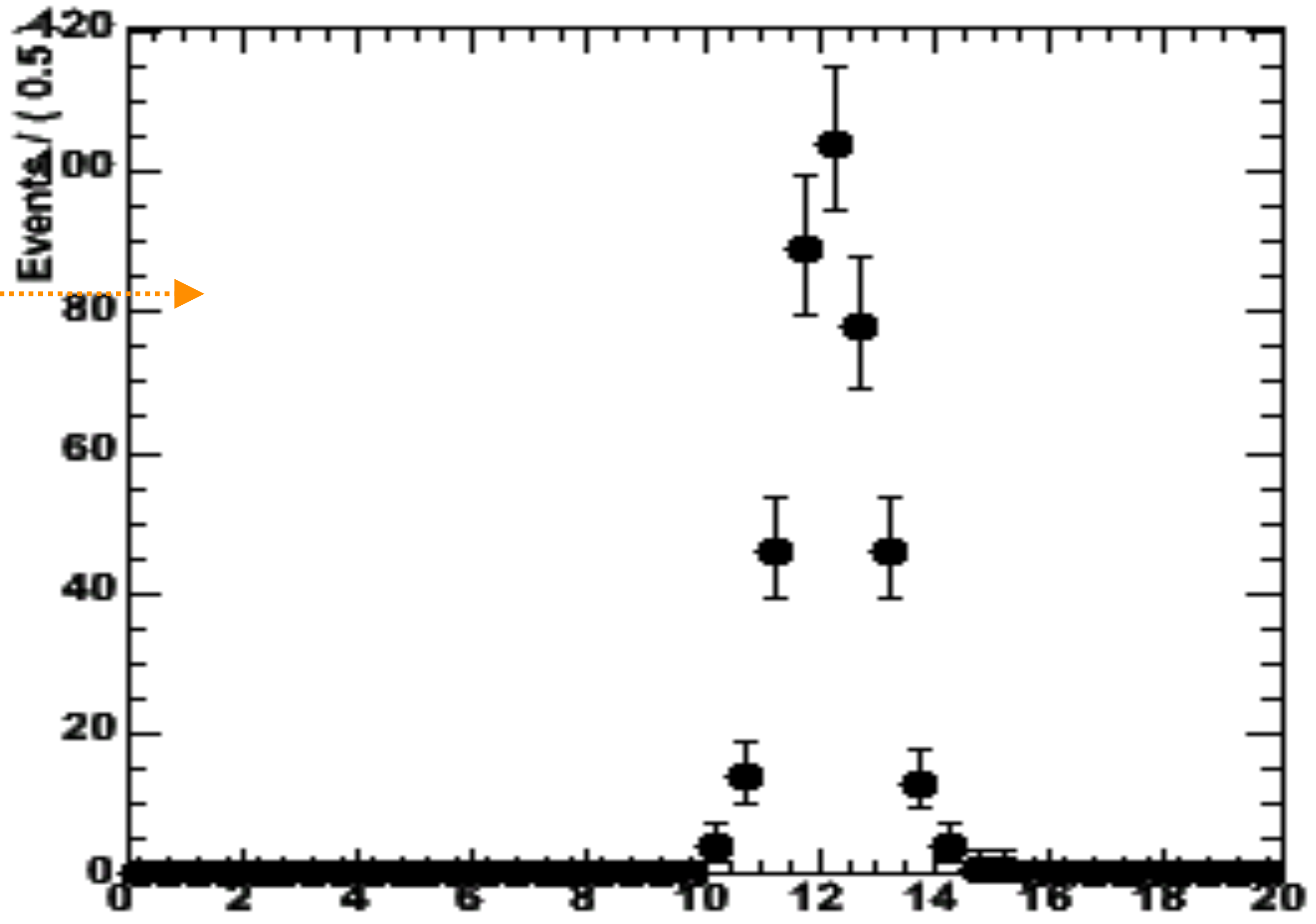


  - Also tools exists to plot residual and pull distributions from curve and histogram in a RooPlot

```
frame->makePullHist();
frame->makeResidHist();
```

Fit Validation Study – The pull distribution

- What about the validity of the error?
  - Distribution of error from simulated experiments is difficult to interpret…
  - We don't have equivalent of N_{sig}(generated) for the error

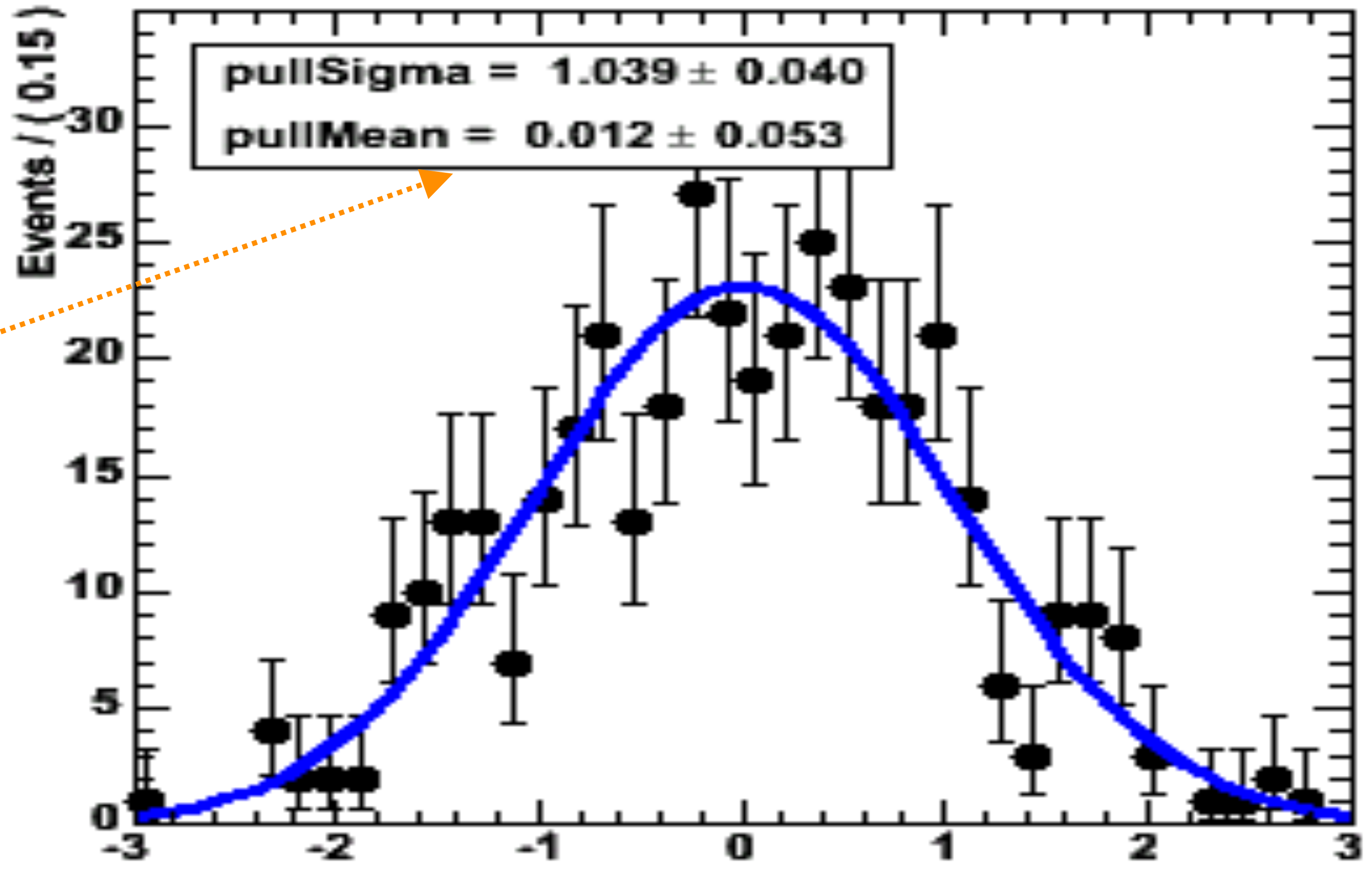


- Solution: look at the ***pull distribution***

Definition:

$$\text{pull}(N_{\text{sig}}) = \frac{N_{sig}^{fit} - N_{sig}^{true}}{\sigma_N^{fit}}$$

  - Properties of pull:
    - **Mean is 0 if there is no bias**
    - **Width is 1 if error is correct**
  - In this example: no bias, correct error within statistical precision of study



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

Next Lecture

- Understand better confidence intervals and hypothesis testing
- See practical examples of estimating frequentist and bayesian intervals using RooStats
  - e.g. show how to make Brazilian plots with RooStat
- See examples of estimating discovery significance