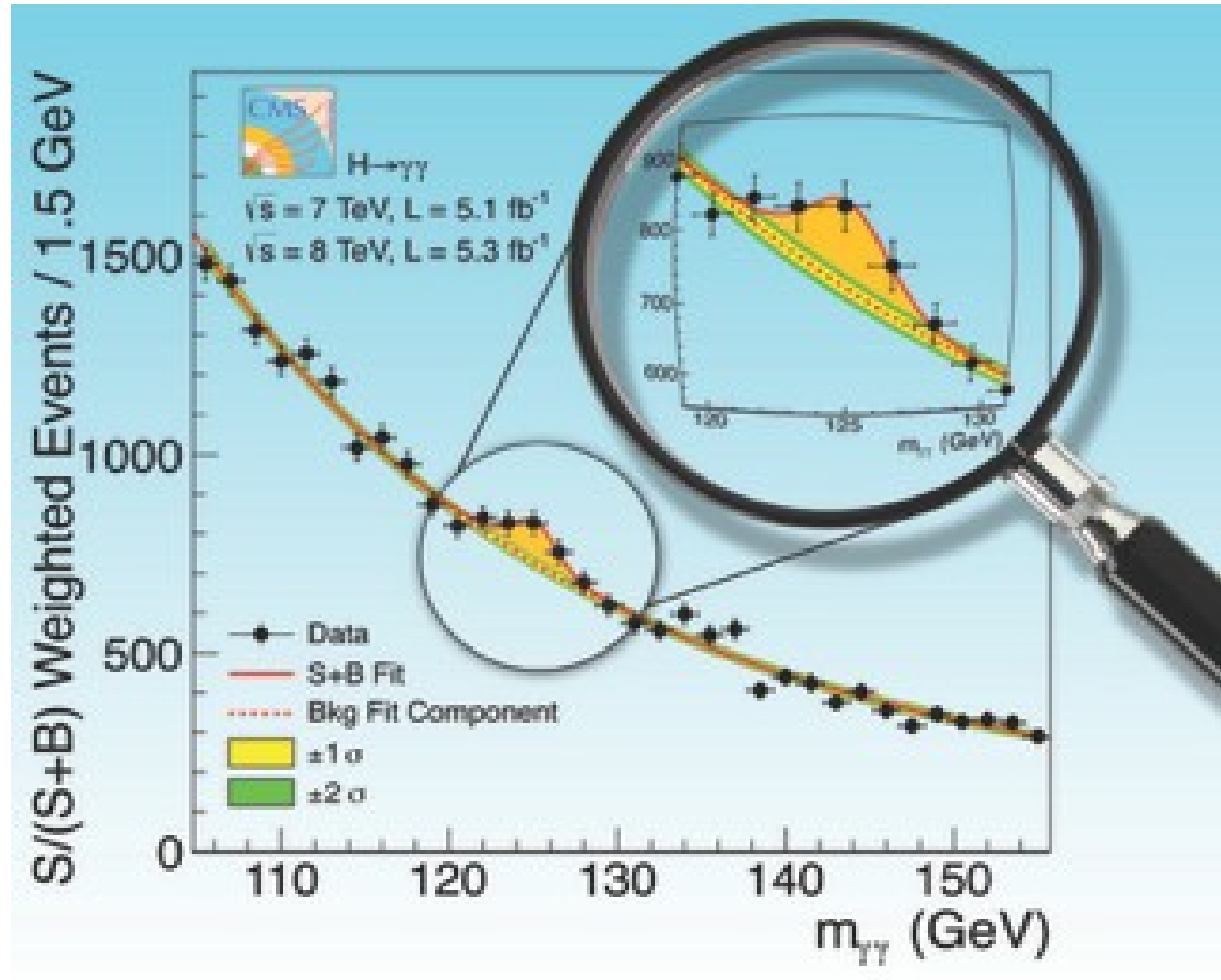


# 第三章：实验统计分析工具ROOT

授课人：董燎原

中国科学院高能物理研究所

# Fitting

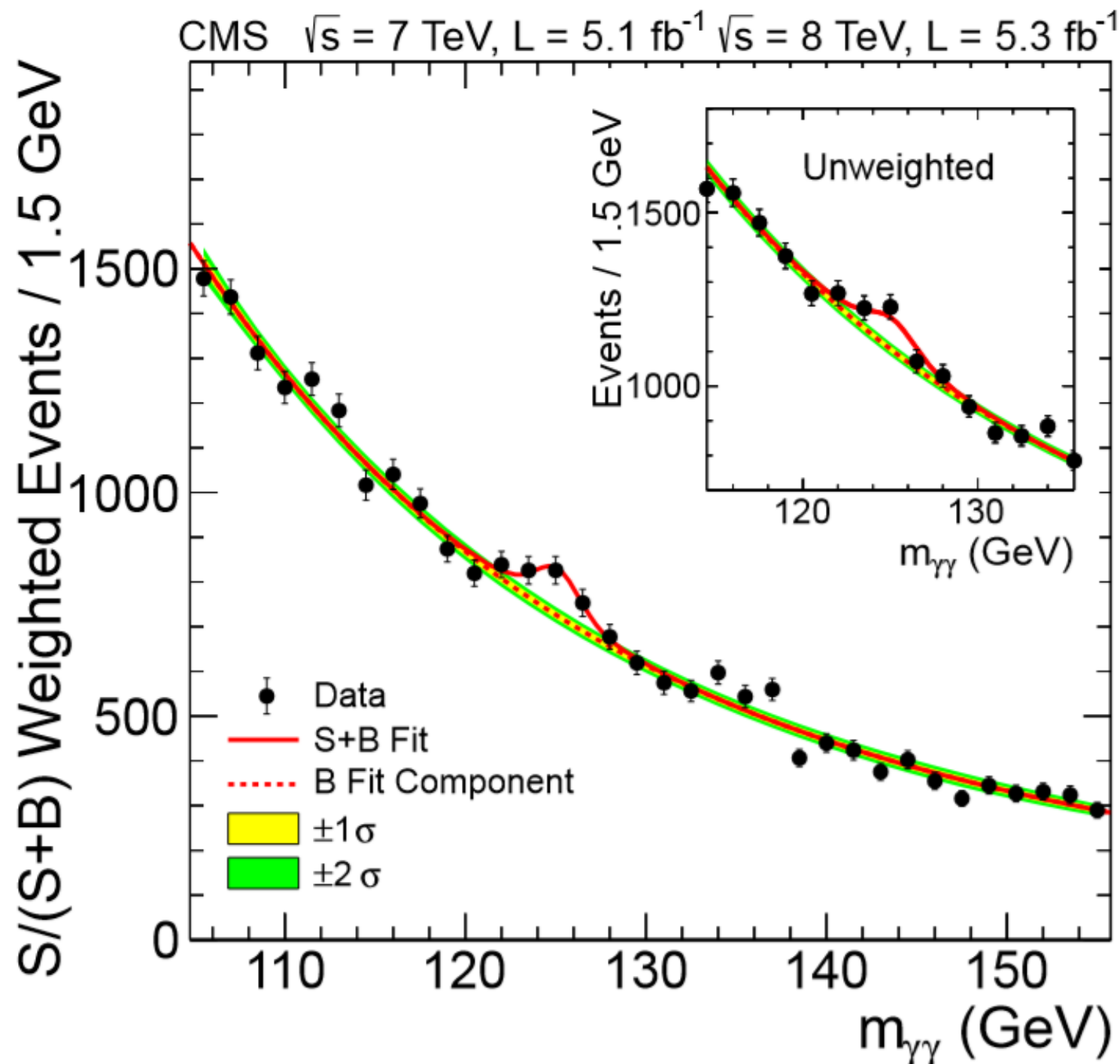


- Introduction to Fitting:
  - what is fitting,
  - how to fit a histogram in ROOT,
  - how to retrieve the fit result.
- Building complex fit functions in ROOT.
- Interface to Minimization.
- Common Fitting problems.
- Using the ROOT Fit GUI (Fit Panel).
- Random number generations in ROOT.
- How to generate random numbers from distributions.

# What is Fitting ?

- **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

( $H \rightarrow \gamma\gamma$ )

Fit for the expected number of Higgs events  
and for the Higgs mass

<https://arxiv.org/pdf/1207.7235.pdf>

# What is Fitting (2)

- A histogram (or a graph) represents an estimate of an underlying distribution (or a function).
- The histogram or the graph can be used to infer the parameters describing the underlying distribution.
- Assume a relation between the observed variables  $y$  and  $x$ :

$$y = f(x, \theta)$$

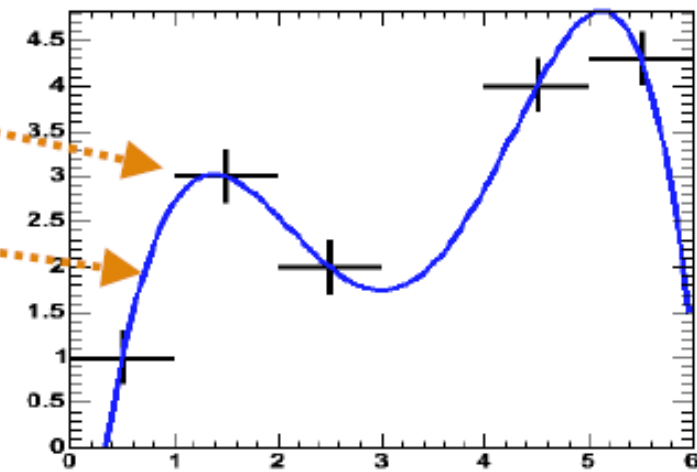
- $f(x, \theta)$  is the fit (model) function,  $\theta$  为待拟合的参数
    - for an histogram  $y$  is the bin content
  - One typically minimizes the deviations between the observed  $y$  and the predicted function values:
    - Least square fit ( $\chi^2$ ) :
      - minimize square deviation
      - weighted by the observed errors
- $\sigma = \sqrt{N}$  for the histograms

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

# A well known estimator – the $\chi^2$ fit

- Given a set of points  $\{(\vec{x}_i, y_i, \sigma_i)\}$  and a function  $f(\mathbf{x}, \mathbf{p})$  define the  $\chi^2$

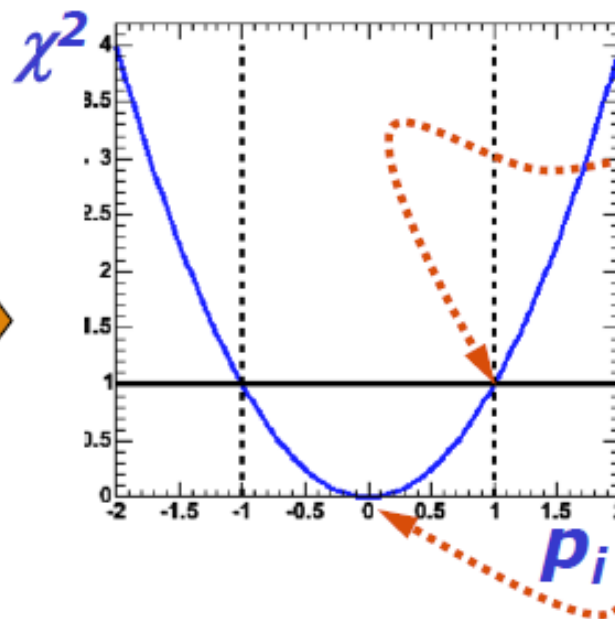
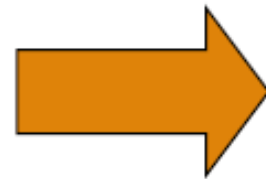
$$\chi^2(\vec{p}) = \sum_i \frac{(y_i - f(\vec{x}; \vec{p}))^2}{\sigma_y^2}$$



- Estimate parameters by minimizing the  $\chi^2(\mathbf{p})$  with respect to all parameters  $p_i$

– In practice, look for

$$\frac{d\chi^2(p_i)}{dp_i} = 0$$



Error on  $p_i$  is given by  $\chi^2$  variation of +1

Value of  $p_i$  at minimum is estimate for  $p_i$

- Well known: but why does it work? Is it always right? Does it always give the best possible error?

# ML Fit of an Histogram

- **Maximum Likelihood (ML) Fit:**

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

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

- The Likelihood for a histogram is obtained by assuming a Poisson distribution in every bin:

- `Poisson( n_obs | n_exp )`

- $n_{\text{obs}}$  is the observed bin content.

- $n_{\text{exp}}$  is the expected bin content, which can be obtained from the fit model function (the underlying distribution of the histogram)

- $n_{\text{exp}} = f(x_c | \theta)$ , where  $x_c$  is the bin center, assuming a linear function within the bin. Otherwise it is obtained from the integral of the function in the bin.

- The least-square fit and the maximum likelihood fit are equivalent when the distribution of observed events in each bin is normal.

- This is true only for large histogram statistics (large bin contents).

- **For low histogram statistics the ML method is the correct one !**

# The Likelihood estimator

- Definition of Likelihood
  - given  $\mathbf{D}(\mathbf{x})$  and  $\mathbf{F}(\mathbf{x};\mathbf{p})$

Functions used in likelihoods must be Probability Density Functions:

$$\int F(\vec{x}; \vec{p}) d\vec{x} \equiv 1, \quad F(\vec{x}; \vec{p}) > 0$$

$$L(\vec{p}) = \prod_i F(\vec{x}_i; \vec{p}), \quad \text{i.e.} \quad L(\vec{p}) = F(x_0; \vec{p}) \cdot F(x_1; \vec{p}) \cdot F(x_2; \vec{p}) \dots$$

- For convenience the *negative log of the Likelihood* is often used

$$-\ln L(\vec{p}) = -\sum_i \ln F(\vec{x}_i; \vec{p})$$

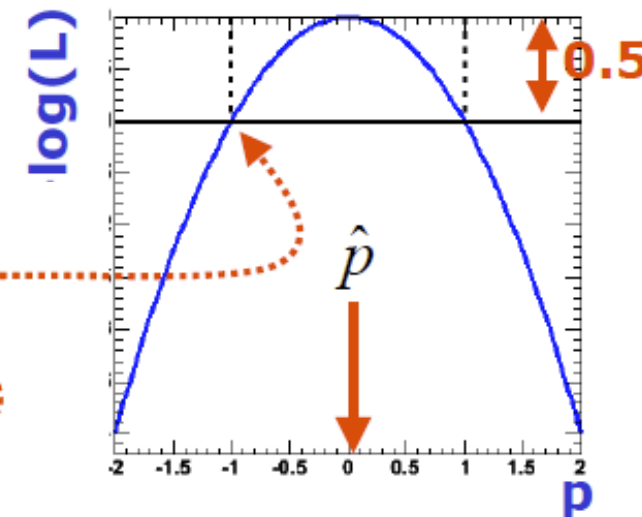
- Parameters are estimated by minimizing  $-\log(L)$ :  $\left. \frac{d \ln L(\vec{p})}{d\vec{p}} \right|_{p_i = \hat{p}_i} = 0$

- Estimator for the parameter variance is  $\hat{\sigma}(p)^2 = \hat{V}(p) = \left( \frac{d^2 \ln L}{d^2 p} \right)^{-1}$

- Visual interpretation of variance estimate

- Taylor expand  $-\log(L)$  around minimum

$$\begin{aligned} \ln L(p) &= \ln L(\hat{p}) + \left. \frac{d \ln L}{dp} \right|_{p=\hat{p}} (p - \hat{p}) + \frac{1}{2} \left. \frac{d^2 \ln L}{d^2 p} \right|_{p=\hat{p}} (p - \hat{p})^2 \\ &= \ln L_{\max} + \left. \frac{d^2 \ln L}{d^2 p} \right|_{p=\hat{p}} \frac{(p - \hat{p})^2}{2} \\ &= \ln L_{\max} + \frac{(p - \hat{p})^2}{2 \hat{\sigma}_p^2} \Rightarrow \ln L(p \pm \sigma) = \ln L_{\max} - \frac{1}{2} \end{aligned}$$





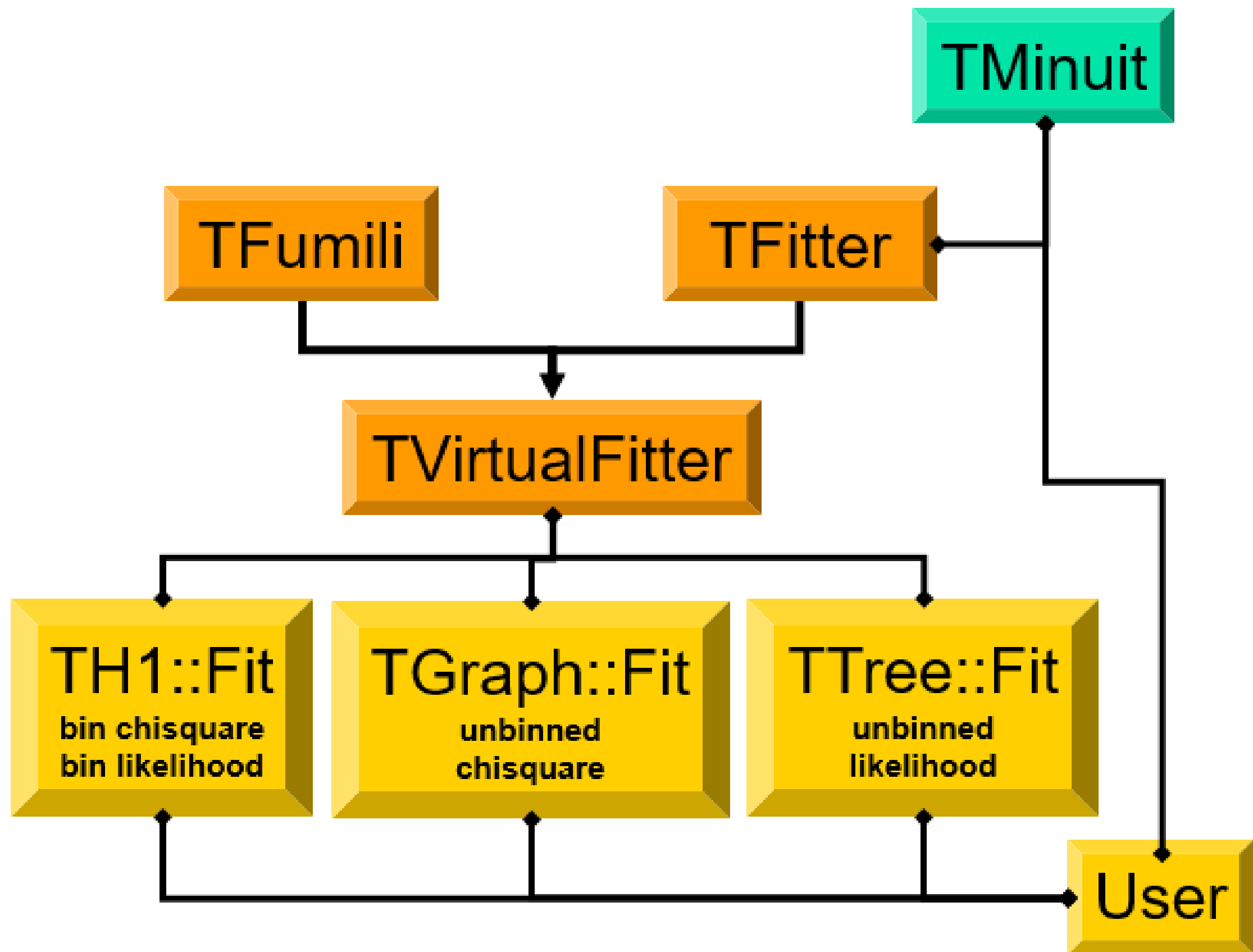
# ML or $\chi^2$ – What should you use?

- $\chi^2$  fit is fastest, easiest
  - Works fine at high statistics
  - Gives absolute goodness-of-fit indication
  - Make (incorrect) Gaussian error assumption on low statistics bins
  - Has bias proportional to  $1/N$
  - Misses information with feature size  $<$  bin size
- Full Maximum Likelihood estimators most robust
  - No Gaussian assumption made at low statistics
  - No information lost due to binning
  - Gives best error of all methods (especially at low statistics)
  - No intrinsic goodness-of-fit measure, i.e. no way to tell if 'best' is actually 'pretty bad'
  - Has bias proportional to  $1/N$
  - Can be computationally expensive for large  $N$
- Binned Maximum Likelihood in between
  - Much faster than full Maximum Likelihood
  - Correct Poisson treatment of low statistics bins
  - Misses information with feature size  $<$  bin size
  - Has bias proportional to  $1/N$

$$-\ln L(p)_{\text{binned}} = \sum_{\text{bins}} n_{\text{bin}} \ln F(\vec{x}_{\text{bin-center}}; \vec{p})$$

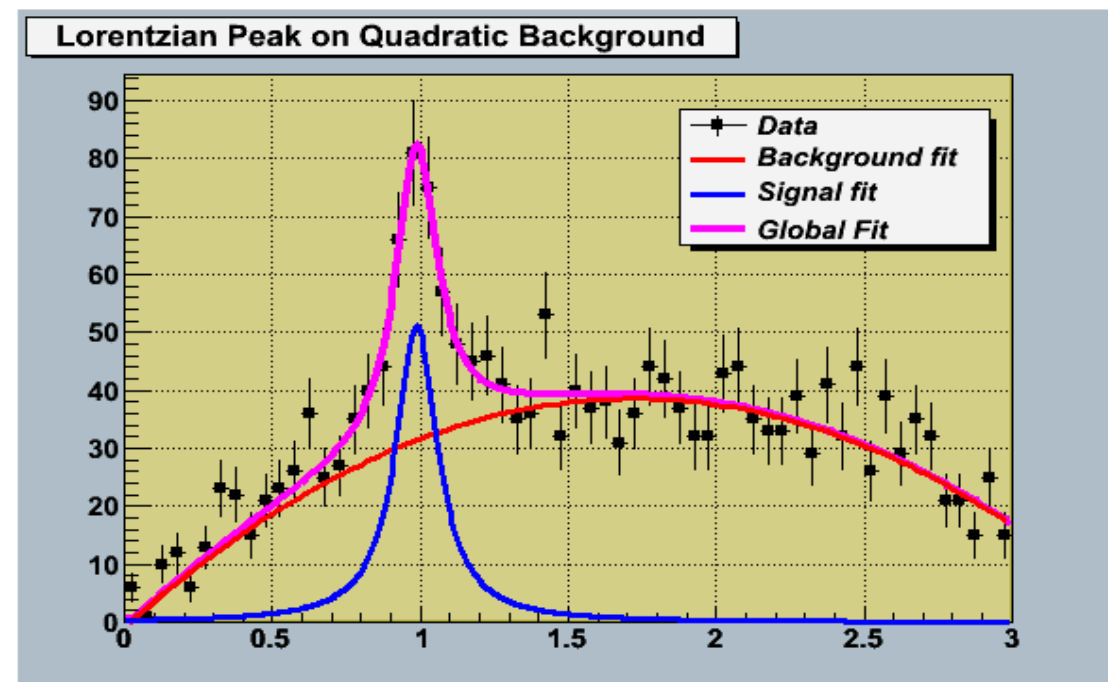
- **How do we do fit in ROOT:**
  - Create first a parametric function object, `TF1`, which represents our model, *i.e.* the fit function.
  - Set the initial values of the function parameters.
  - Fit the data object (Histogram or Graph):
    - call the `Fit` method on the Histogram or Graphs passing 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.

# Fitting



# Fitting - Interface

- Minimization packages: **Minuit** and **Fumili**
- Fitting can be done through the general interface of
  - **TH1::Fit (binned data)**  
Chisquare and Loglikelihood methods
  - **TGraph::Fit**  
unbinned data
  - **TGraphErrors::Fit**  
data with errors
  - **TGraphAsymmErrors::Fit**  
taking into account asymmetry of errors
  - **TTree::Fit** and **TTree::UnbinnedFit**
- **RooFit** for object-oriented data modeling (下节课介绍).  
---- Distributed with ROOT starting from version 5.02-00



# the Fit Method

- Use the **TH1::Fit** method

```
void Fit(const char *fname, Option_t *option, Option_t *goption,  
        Axis_t xmin, Axis_t xmax)
```

- **\*fname** - the name of the fitted function. This name may be one of ROOT pre-defined function names or a user-defined function.  
Predefined functions:
  - ✓ **gaus**: Gaussian function with 3 parameters:  $p_0 \cdot \exp(-0.5 \cdot ((x-p_1)/p_2)^2)$
  - ✓ **expo**: an Exponential with 2 parameters:  $\exp(p_0 + p_1 \cdot x)$
  - ✓ **polN**: a polynomial of degree N:  $p_0 + p_1 \cdot x + p_2 \cdot x^2 + \dots$
  - ✓ **landau**: Landau function with mean and sigma.
- **\*option** - fitting option
- **\*goption** - graphics option which is the same as in the **TH1::Draw()**
- **xmin, xmax** - specify the range over which to apply the fit.

# the Fitting Option

"W"	Set all weights to 1 for non empty bins; ignore error bars
"WW"	Set all weights to 1 including empty bins; ignore error bars
"I"	Use integral of function in bin instead of value at bin center
"L"	Use log likelihood method (default is chi-square method)
"U"	Use a user specified fitting algorithm
"Q"	Quiet mode (minimum printing)
"V"	Verbose mode (default is between Q and V)
"E"	Perform better errors estimation using the Minos technique
"M"	Improve fit results
"R"	Use the range specified in the function range
"N"	Do not store the graphics function, do not draw
"0"	Do not plot the result of the fit. By default the fitted function is drawn unless the option "N" above is specified.
"+"	Add the new fitted function to the list of fitted functions (by default, the previous function is deleted and only the last one is kept)
"B"	Use this option when you want to fix one or more parameters and the fitting function is like polN, expo, landau, gaus.
"LL"	An improved Log Likelihood fit in case of very low statistics and when bin contents are not integers.
"C"	In case of linear fitting, don't calculate the chisquare (saves time).
"F"	If fitting a polN, switch to Minuit fitter (by default, polN functions are fitted by the linear fitter).

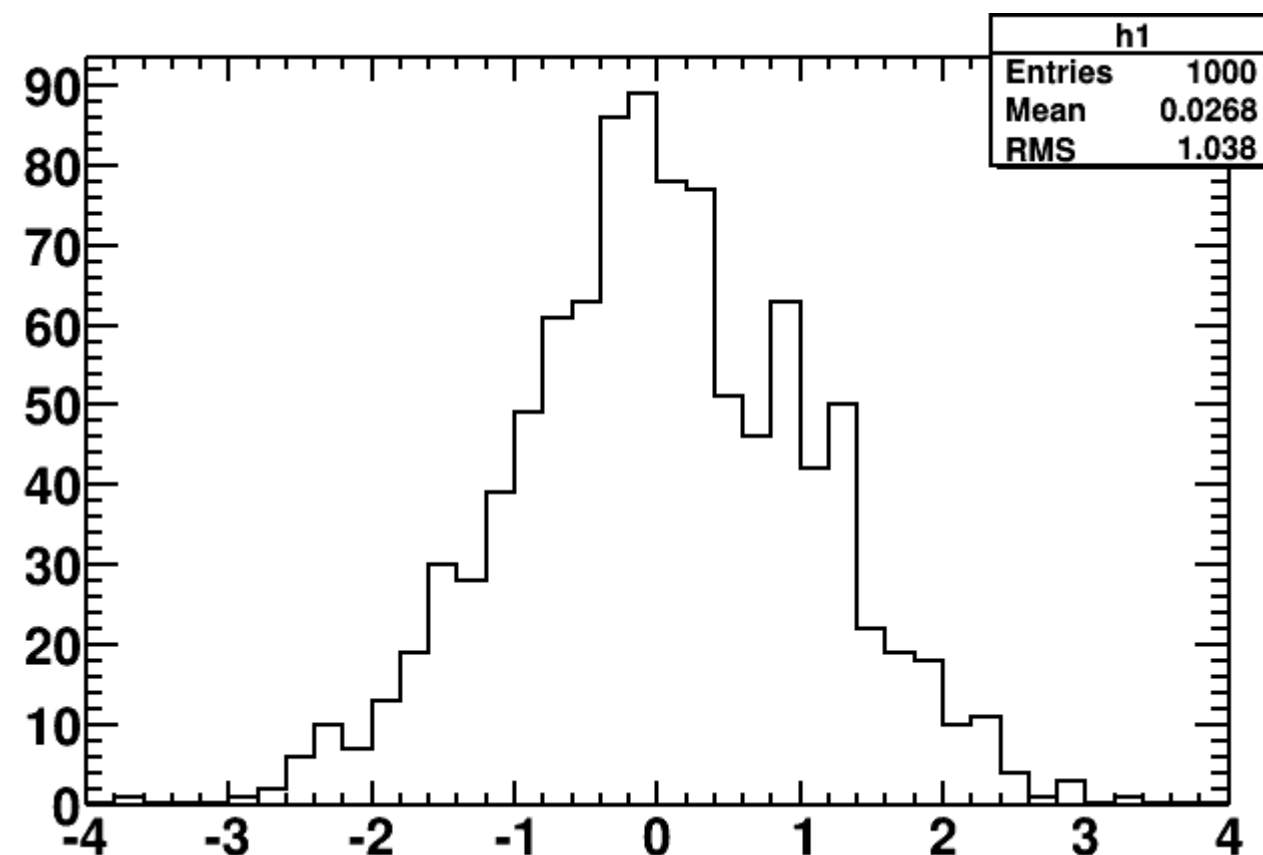
- ▣ Empty bins are excluded in the fit when using the Chi-square fit method. When fitting the histogram with the low statistics, it is recommended to use the **Log-Likelihood** method (**RooFit** is recommended!!!).

# Simple Gaussian Fitting

- Recalling our previous histogram:
  - suppose we do not know how it was generated;
  - we want to estimate the mean and sigma of the underlying gaussian distribution.

root draw.C

```
#include "TH1.h"
#include "TF1.h"
#include "TRandom3.h"
void p(){
  TH1D * h1 = new TH1D("h1","Example",40,-4.,4.);
  for (int i = 0; i < 1000; ++i) {
    double x = gRandom->Gaus(0,1);
    h1->Fill(x);
  }
  h1->Draw();
}
```



# 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])");
```

- or we can use pre-defined functions defined in `TFormula` (see `TFormula` documentation for the list of them):

```
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.
- We will see later in general how to build a more complex function objects
  - e.g. by using other functions



# Fitting Histogram

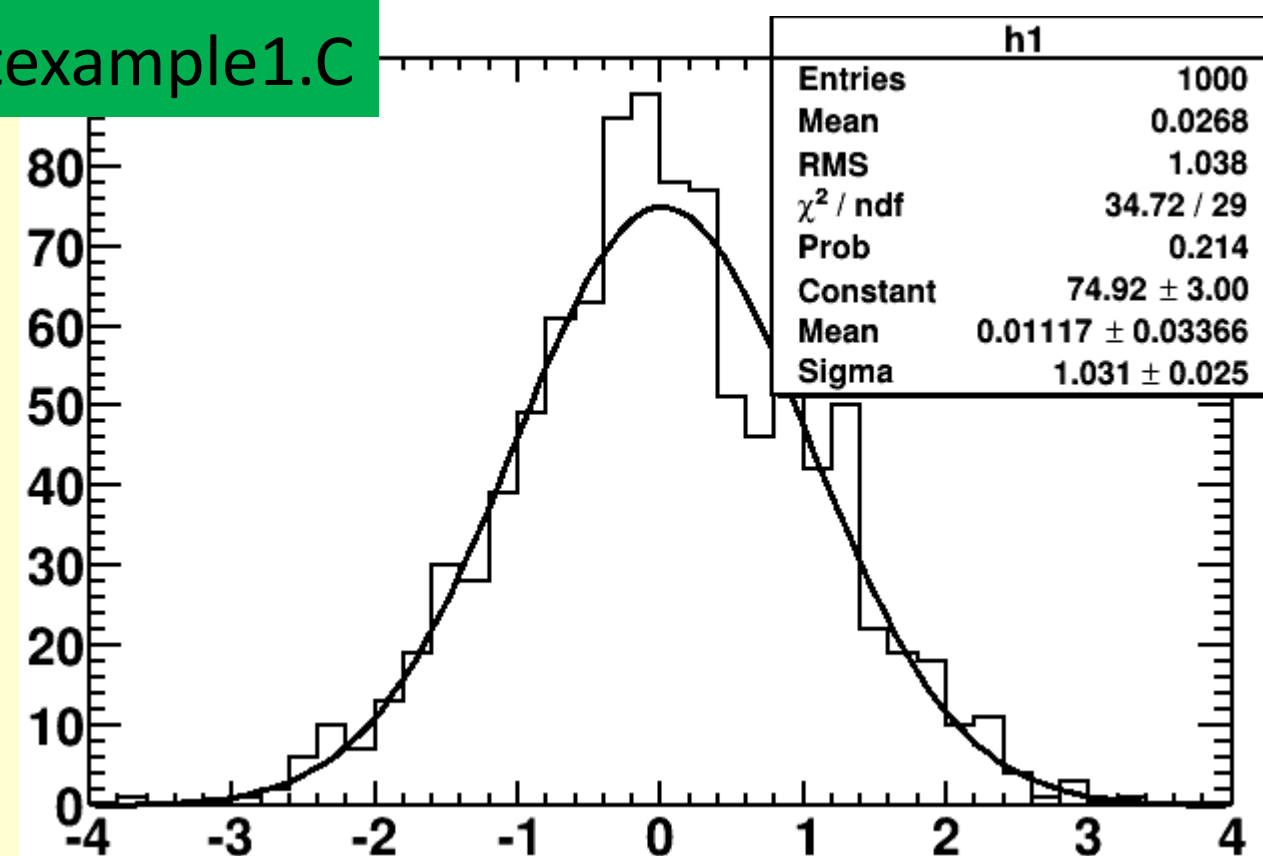
- How to fit the histogram (use a predefined function):
  - after creating the function one needs to set the initial value of the parameters
  - then we can call the `Fit` method of the histogram class

```
#include "TMath.h"
#include "TH1.h"
#include "TF1.h"
#include "TRandom3.h"
#include "TStyle.h"

void fitexample1() {
    TH1D * h1 = new TH1D("h1", "Example", 40, -4., 4.);
    for (int i = 0; i < 1000; ++i) {
        double x = gRandom->Gaus(0,1);
        h1->Fill(x);
    }

    TF1 *func= new TF1("func", "gaus");
    func->SetParameters(1,0,1);
    // set the parameters to 1, 0, 1
    gStyle->SetOptFit(1111);
    h1->Fit("func");
}
```

运行:  
root fitexample1.C



For displaying the fit parameters:

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

可用下面的命令编译，以检查错误：  
root [1] .L fitexample1.C+

# Fitting Histogram (2)

- How to fit the histogram (use a user-defined Function):

```
#include "TMath.h"
#include "TH1.h"
#include "TF1.h"
#include "TRandom3.h"
#include "TStyle.h"

// define a function with 3 parameters
Double_t fitf(Double_t *x, Double_t *par) {
    Double_t arg = 0;
    if (par[2] != 0) arg = (x[0] - par[1])/par[2];
    Double_t fitval = par[0]*TMath::Exp(-0.5*arg*arg);
    return fitval;
}

void fitexample2() {
    TH1D * h1 = new TH1D("h1", "Example", 40, -4., 4.);
    for (int i = 0; i < 1000; ++i) {
        double x = gRandom->Gaus(0,1);
        h1->Fill(x);
    }
    TF1 *func= new TF1("func", fitf, -3, 3, 3);
    // set the parameters to mean and RMS of the histogram
    func->SetParameters(500, h1->GetMean(), h1->GetRMS());
    // give the parameters meaningful names
    func->SetParNames ("Constant", "Mean_value", "Sigma");
    gStyle->SetOptFit(1111);
    h1->Fit("func");
}
```

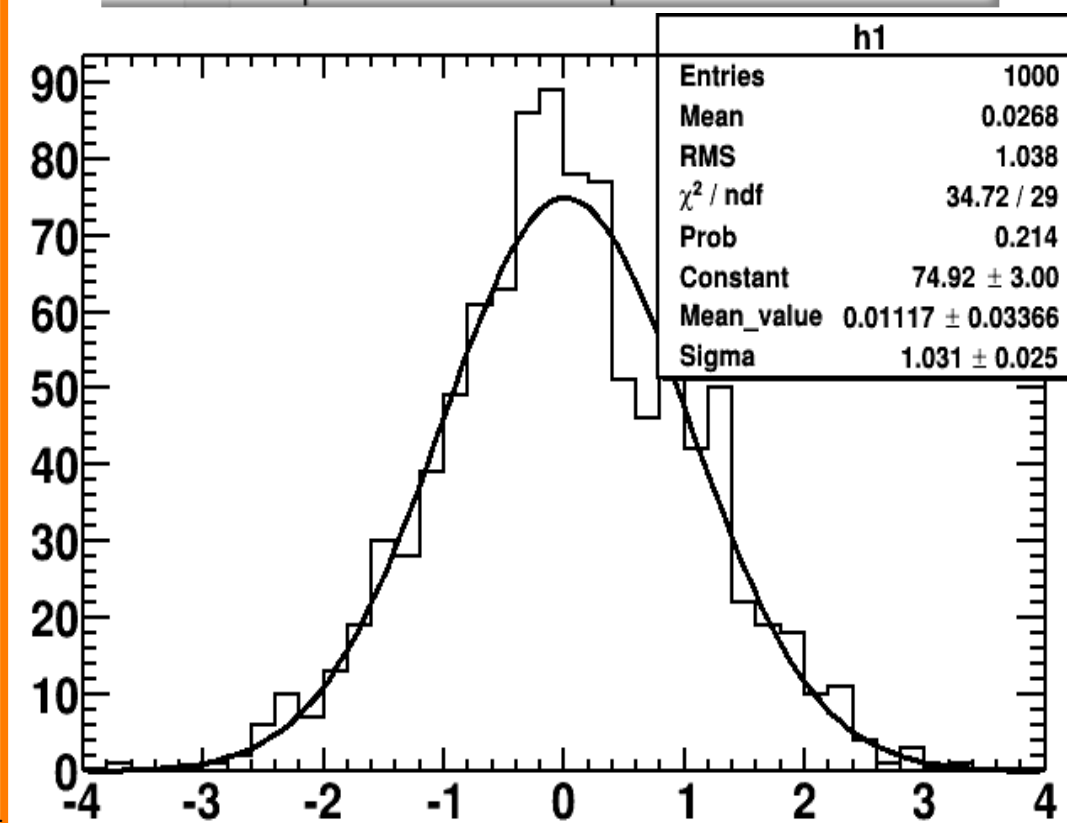
\$ROOTSYS/tutorials/fit/myfit.C

For displaying the fit parameters:

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

four digits: mode = pcev ( default = 0111 )

- ☐ p = 1 print probability
- ☐ c = 1 print Chi-square/number of degrees of freedom
- ☐ e = 1 print errors (if e=1, v must be 1)
- ☐ v = 1 print name/values of parameters



TF1 \* f = new  
TF1("f", fobj, xmin, xmax, npar);  
// create TF1 class with n-parameters and  
range [xmin, xmax]

```
root [0] TMath::Prob(34.72, 29)
(Double_t)2.13881477761421218e-01
```

# Fixing and Setting Parameters

- Parameters must be initialized to some value as close as possible to the expected values before invoking the Fit method.

- ▣ To set bounds for one parameter, use **TF1::SetParLimits**:

```
func->SetParLimits(0, -1, 1); //parameter 0 varies from -1 to 1  
func->SetParameter(4, 10); //initialize parameter 4 to 10  
func->SetParLimits(4, 10, 10); //parameter 4 is fixed
```

- ▣ To fix a parameter to 0, one must call the FixParameter function:

```
func->SetParameters(3.1, 1.e-6, -1.5, 0, 100);  
func->SetParLimits(3, -10, 4);  
func->FixParameter(4, 0);
```

# 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);
```

- The Chisquare:

```
fitFunc->GetChisquare();
```

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

```
Minimizer is Minuit / Migrad
Chi2      =      34.7164
NDF       =      29
Edm       = 1.94352e-08
NCalls    =      61
Constant  = 74.9237 +/- 3.00474
Mean      = 0.0111669 +/- 0.0336562
Sigma     = 1.03102 +/- 0.0253614
root [18]
```

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

# Access to the Fit Covariance Matrix

## Example1:

```
TH1F h("h", "test", 100, -2, 2);  
h.FillRandom("gaus", 1000);  
h.Fit("gaus");  
Double_t matrix[3][3];  
gMinuit->mneumat(&matrix[0][0], 3);
```

## Example2:

```
TH1F h("h", "test", 100, -2, 2);  
h.FillRandom("gaus", 1000);  
h.Fit("gaus");  
TVirtualFitter *fitter = TVirtualFitter::GetFitter();  
TMatrixD matrix(3, 3, fitter->GetCovarianceMatrix());  
Double_t errorFirstPar = fitter->GetCovarianceMatrixElement(0,0);
```

# Some Fitting Options

- Fitting in a Range

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

- Fitting more functions to an histogram (option "+")

- `h1->Fit("expo","+","", 2., 4);`

- Quite / Verbose:

- option "Q"/"V".

- Likelihood fit:

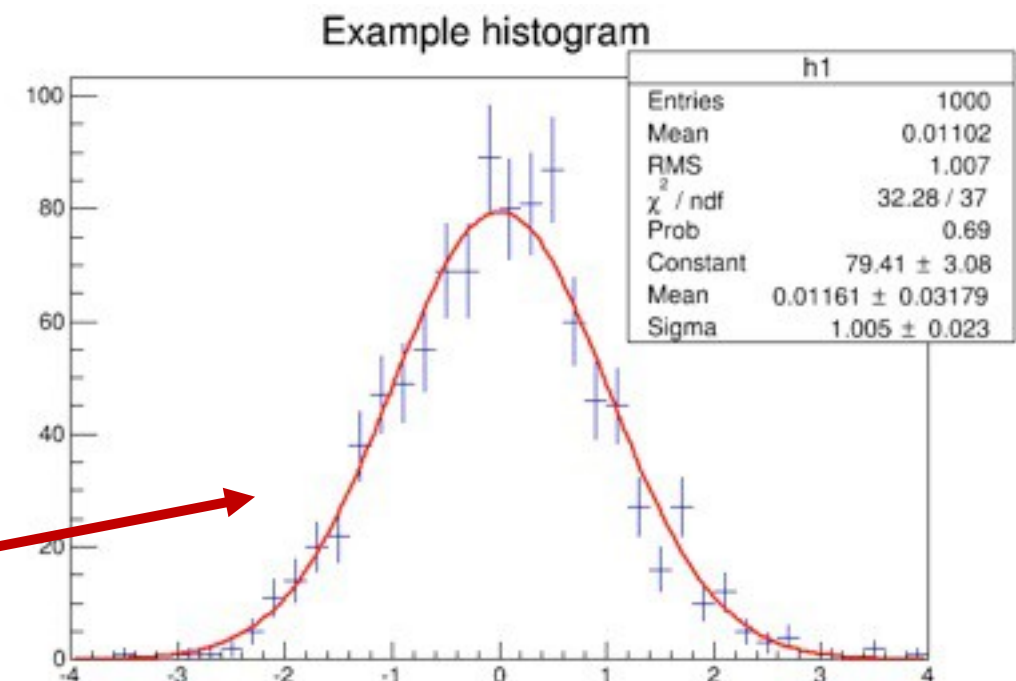
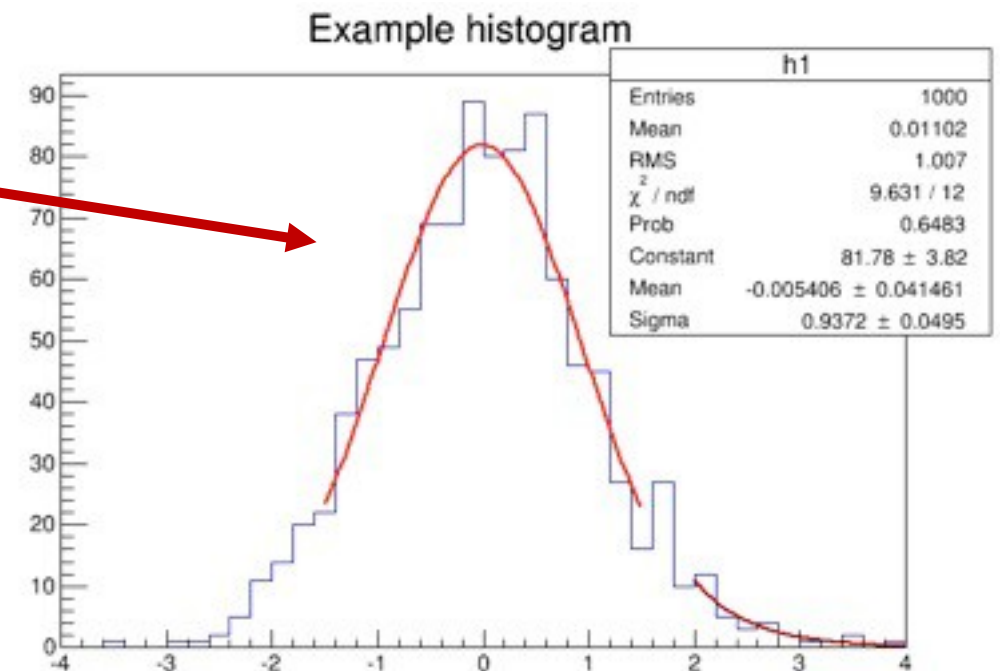
- option "L" for count histograms;
  - option "LW" in case of weighted counts.

- Return a fit result class:

- option "S"

- Plotting options for the histogram can be passed as well:

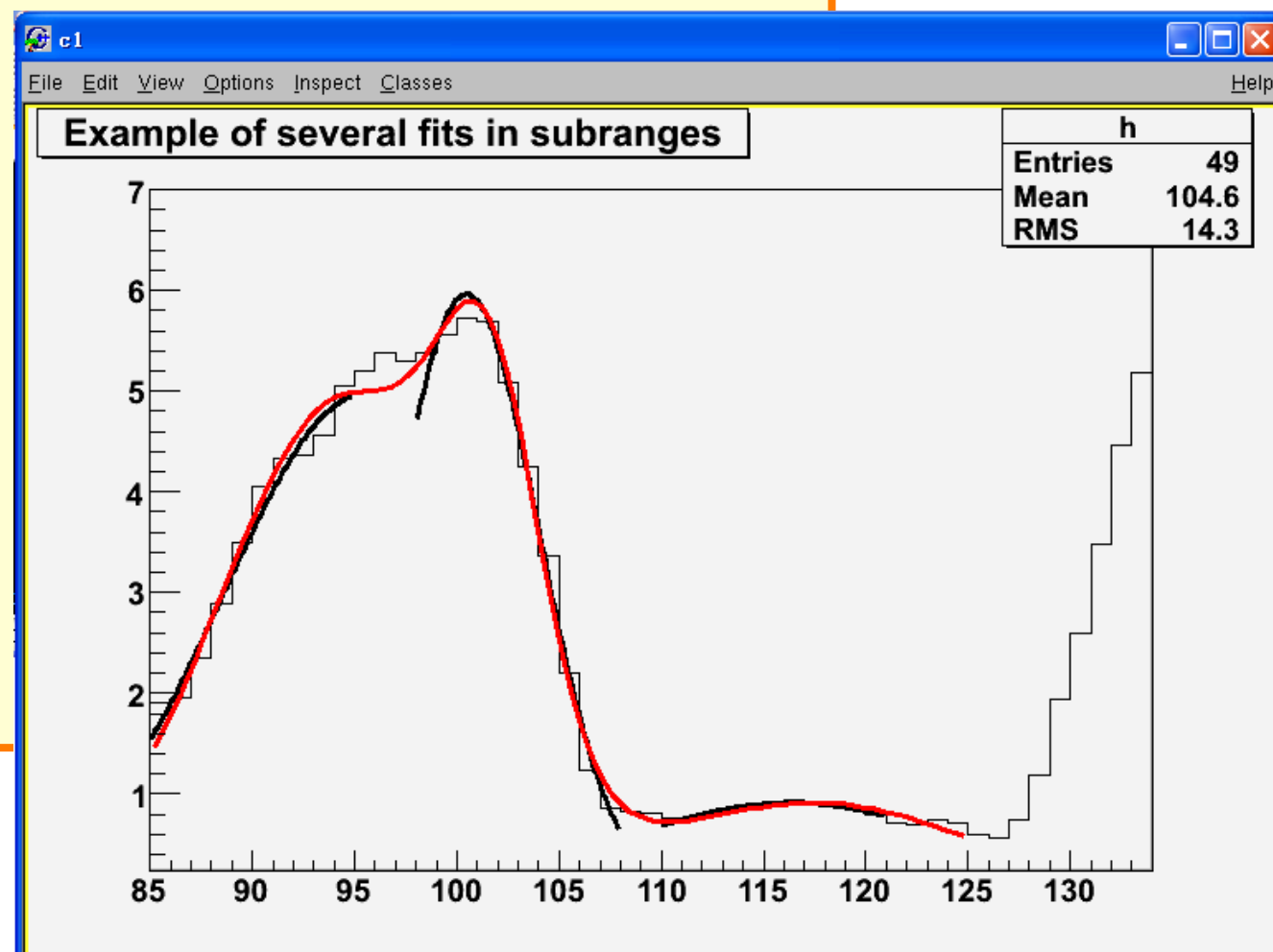
- `h1->Fit("gaus","L","E");`



# Fitting Multiple Sub Ranges

```
g1 = new TF1("m1","gaus",85,95);
g2 = new TF1("m2","gaus",98,108);
g3 = new TF1("m3","gaus",110,121);
// The total is the sum of the three, each has 3 parameters
total = new TF1("mstotal","gaus(0)+gaus(3)+gaus(6)",85,125);
// Create a histogram and set it's contents
h = new TH1F("g1","Example",np,85,134);
h->SetMaximum(7);
for (int i=0; i<np; i++) h->SetBinContent(i+1,x[i]);
// Define the parameter array
//for the total function
Double_t par[9];
// Fit each function
h->Fit(g1,"R");
h->Fit(g2,"R+");
h->Fit(g3,"R+");
// Get the parameters from the fit
g1->GetParameters(&par[0]);
g2->GetParameters(&par[3]);
g3->GetParameters(&par[6]);
// Use the parameters on the sum
total->SetParameters(par);
h->Fit(total,"R+");
```

*complete code in \$ROOTSYS/tutorials/fit/multifit.C*





# Combining Functions

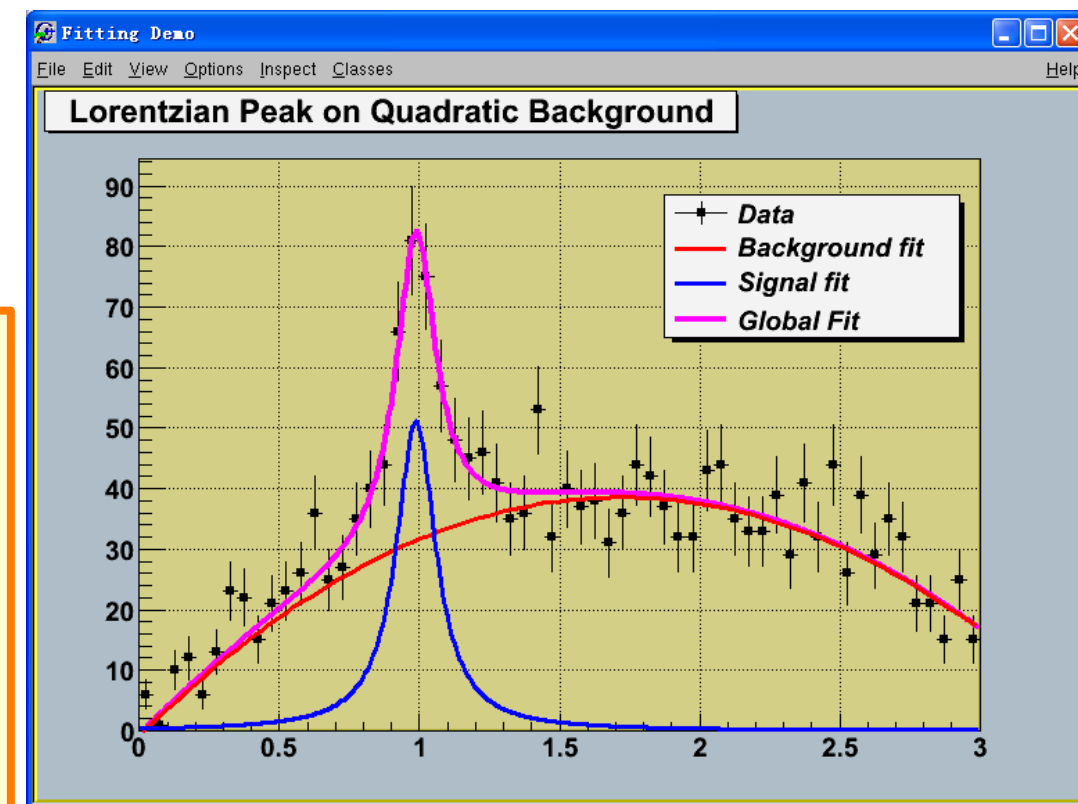
- The combination function of a background and Lorentzian peak. Each function contributes 3 parameters:

$$y(E) = a_1 + a_2 E + a_3 E^2 + \frac{A_p \left( \frac{G}{2\pi} \right)}{(E - m)^2 + \left( \frac{G}{2} \right)^2}$$

```
// Quadratic background function
Double_t background(Double_t *x, Double_t *par) {
return par[0]+par[1]*x[0]+par[2]*x[0]*x[0];
}

// Lorentzian Peak function
Double_t lorentzianPeak(Double_t *x, Double_t *par){
return (0.5*par[0]*par[1]/TMath::Pi()) /
TMath::Max(1.e-10, (x[0]-par[2])*
(x[0]-par[2])+.25*par[1]*par[1]));
}

// Sum of background and peak function
Double_t fitFunction(Double_t *x, Double_t *par) {
return
background(x,par) + lorentzianPeak(x,&par[3]);
}
```



par[0]:  $A_p$   
par[1]:  $G$   
par[2]:  $m$



# Combining Function Fit and Plot

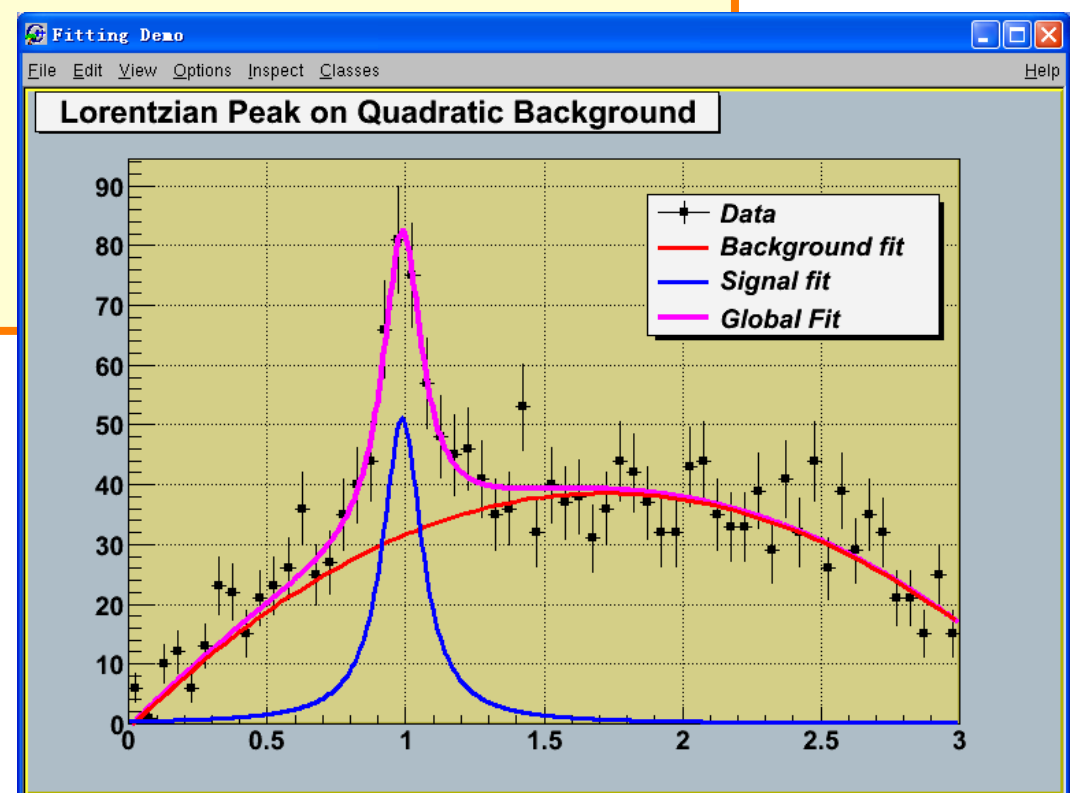
*\$ROOTSYS/tutorials/fit/FittingDemo.C*

```
TH1F *histo = new TH1F("example_9_1","Lorentzian",
60,0,3);
TF1 *fitFcn = new TF1("fitFcn",fitFunction,0,3,6);
// first set each parameter to 1
fitFcn->SetParameters(1,1,1,1,1,1);
histo->Fit("fitFcn","0");
// second try: set start values for some parameters
fitFcn->SetParameter(4,0.2); // width
fitFcn->SetParameter(5,1); // peak
histo->Fit("fitFcn","V+","ep");

...
TF1 *backFcn = new TF1("backFcn",background,0,3,3);
TF1 *signalFcn = new TF1("signalFcn",lorentzianPeak,0,3,3);
Double_t par[6];
// writes the fit results into the par array
fitFcn->GetParameters(par);
backFcn->SetParameters(par);
backFcn->Draw("same");
signalFcn->SetParameters(&par[3]);
signalFcn->Draw("same");
```

运行

```
root FittingDemo.C
root> .x FittingDemo.C
root> .x FittingDemo.C+
```



# Building More Complex Functions

- It is possible to write some complex formulae and pass as string in the constructor of `TF1`
  - but difficult and prone to error
- Better to write directly the functions in C/C++
- A parametric `TF1` can be constructed from
  - a general free function with parameters:

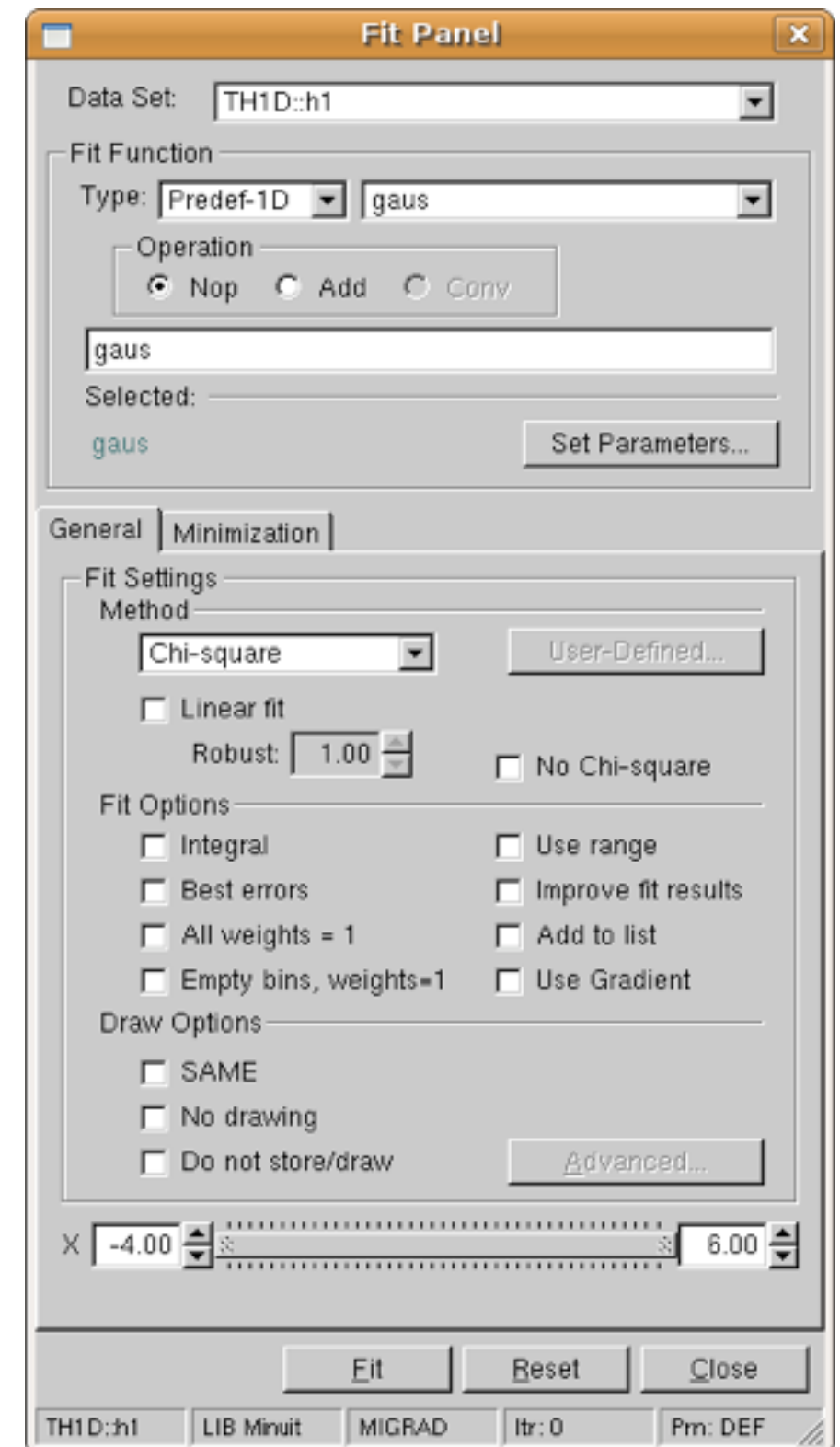
```
double function(double *x, double *p){  
    return p[0]*TMath::Gaus(x[0],p[0],p[1]);  
}  
  
TF1 * f1 = new TF1("f1",function,xmin,xmax,npar);
```

- 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");
```

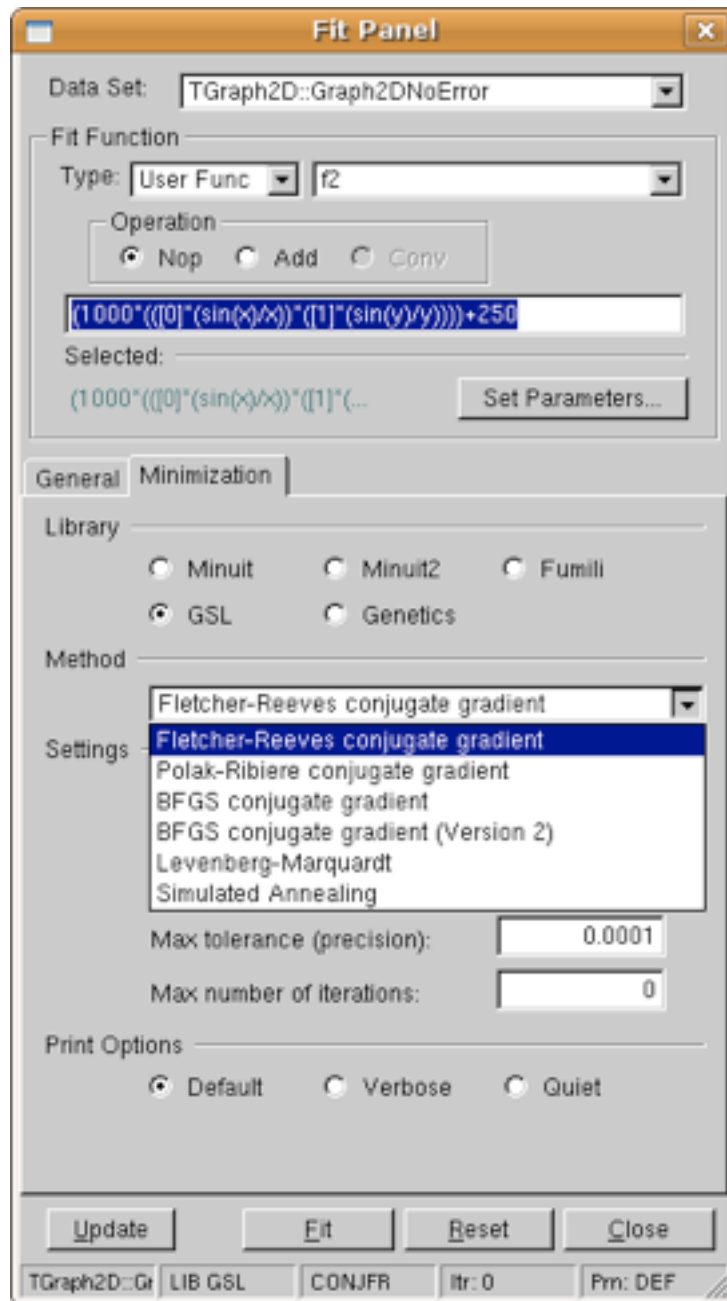
# The Fit Panel

- 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

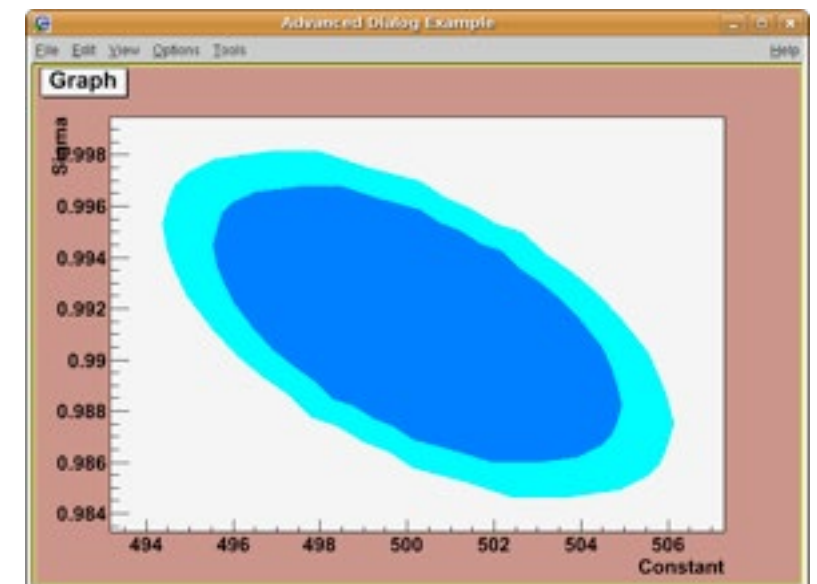
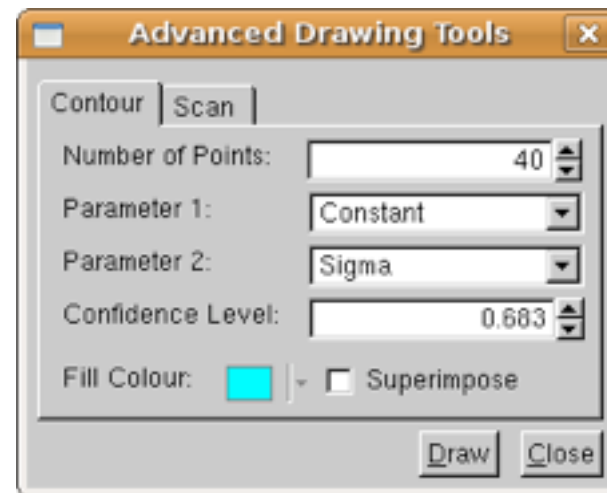


# Fit Panel (2)

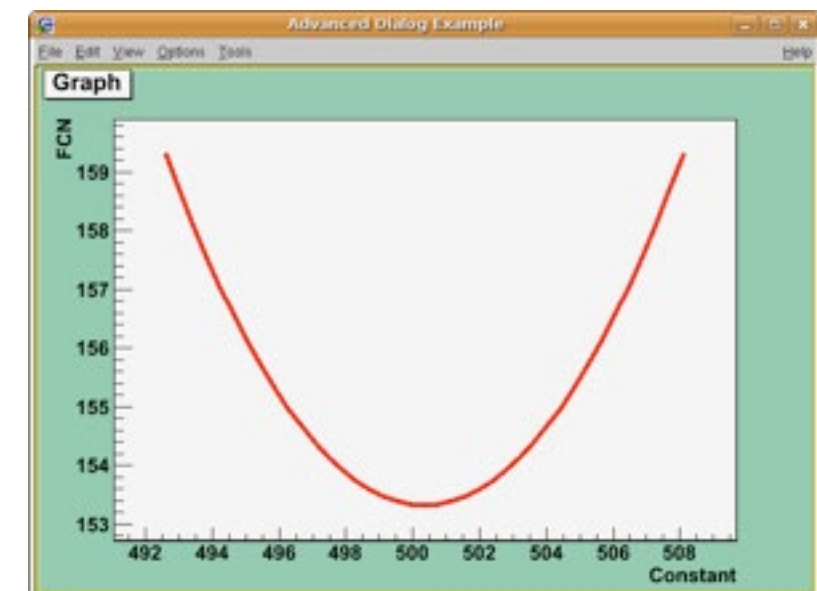
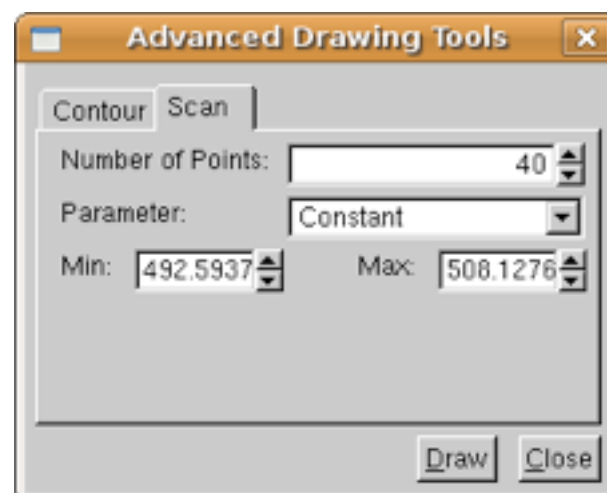
- The Fit Panel provides also extra functionality:
  - Control the minimization
  - Advanced drawing tools



Contour plot



Scan plot of minimization function



# Access to the Fitter Information

- ❑ **TH1::Fit** function calls the abstract fitter **TVirtualFitter**. The default fitter is **TFitter** (calls **TMinuit**).
- ❑ The default fitter can be set via **TVirtualFitter::SetDefaultFitter**. For example, to call the "**Fumili**" fitter:

```
TVirtualFitter::SetDefaultFitter("Fumili");
```

- ❑ During the fitting process, the objective function: **chisquare**, **likelihood** or any user-defined algorithm is called.

```
TVirtualFitter *fitter = TVirtualFitter::GetFitter();//the current fitter
TH1 *hist = (TH1*)fitter->GetObjectFit(); //the histogram being fitted
TF1 *f1 = (TF1*)fitter->GetUserFunction(); //the user theoretical function
```

- ❑ By default, the fitter **TMinuit** is initialized with a maximum of 25 parameters.

# 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:
  - Other commands are also available to control the minimization:

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

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

# Minimization Algorithm

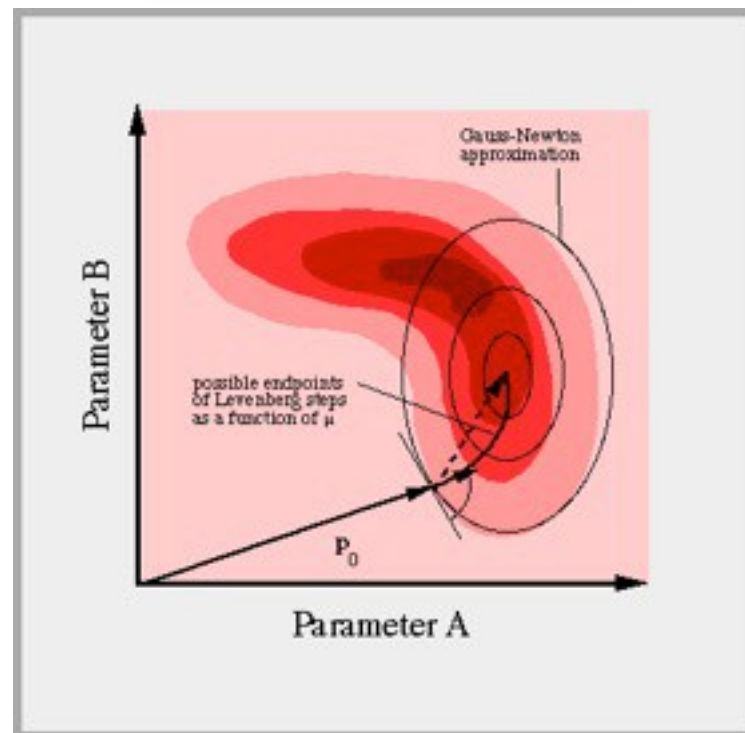
- **MIGRAD**: almost applicable to all the functions, but heavily reliant on the knowledge of first derivative
- **MINIMIZE**: equivalent to MIGRAD, except to call SIMPLEX when MIGRAD fails
- **SCAN**: scan one parameter at a time
- **SEEK**: a Monte-Carlo search for minima (nearly obsolete)
- **SIMPLEX**: multi-dimensional, robust precision, slow, not reliable error matrix



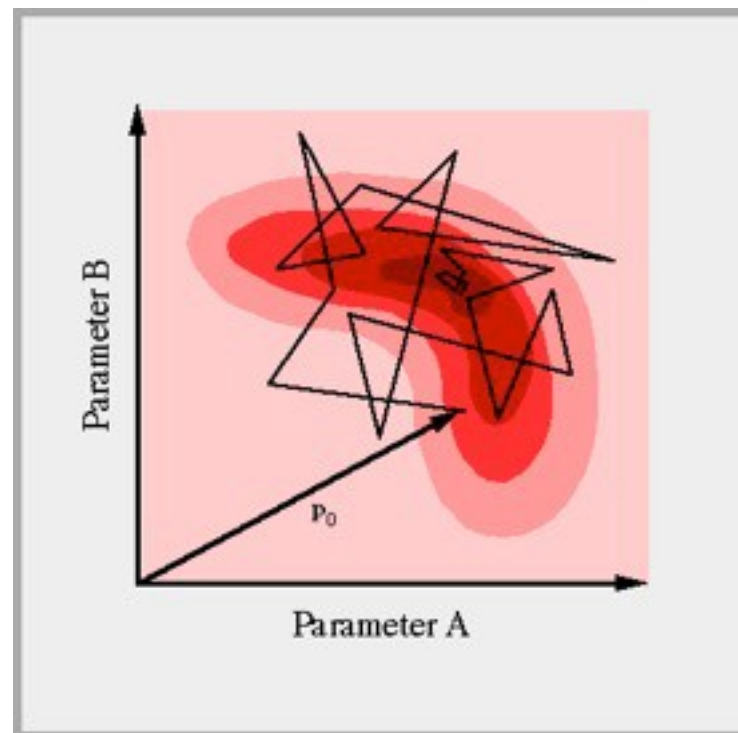
# Minimization Techniques

- 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.

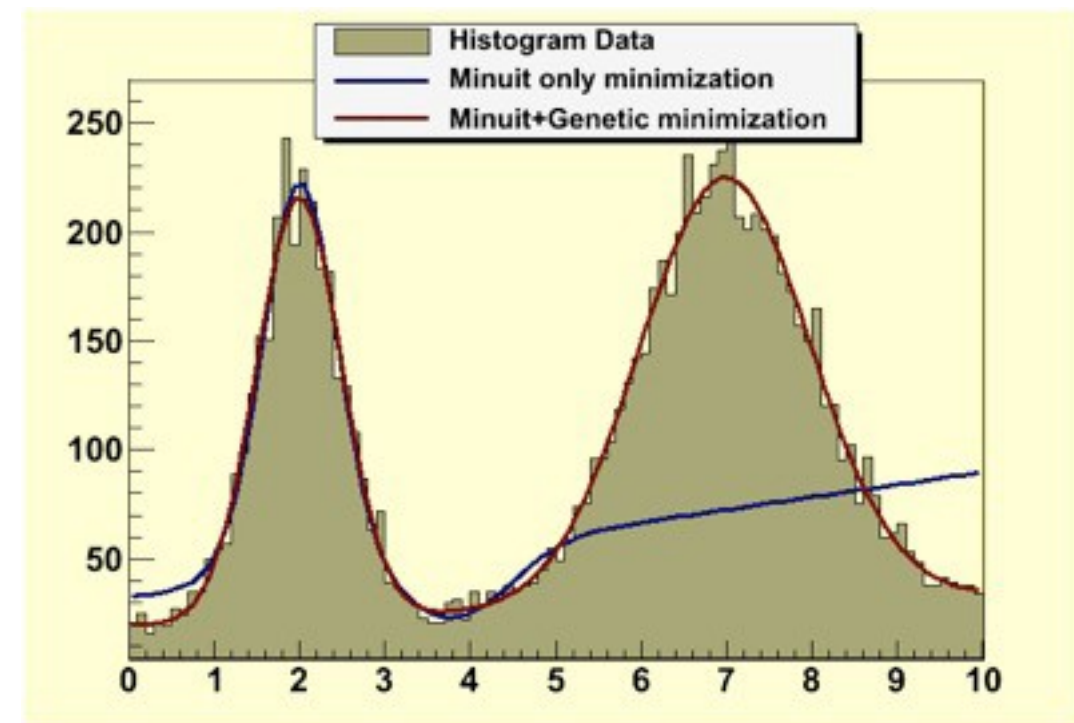
Quadratic Newton



Simulated Annealing



Example: Fitting 2 peaks in a spectrum





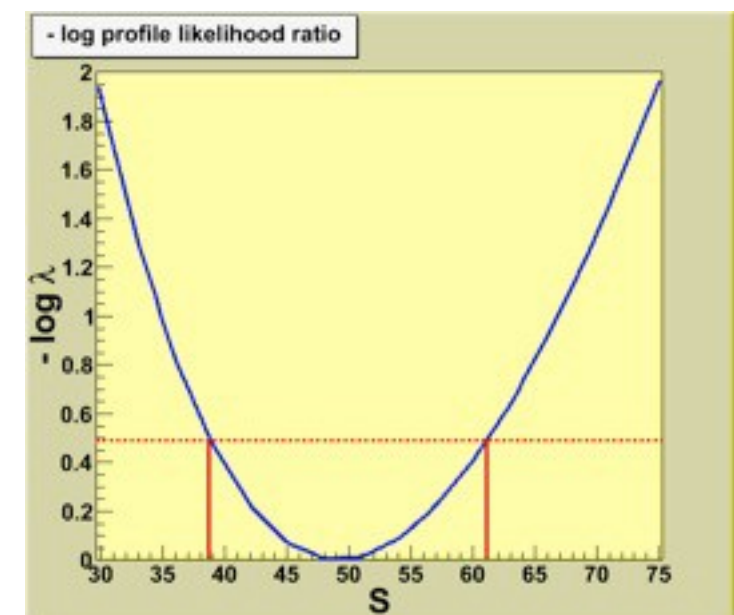
# Parameter Errors

- 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);
```



# Interface to Minimization

- A common interface for all ROOT Minimizer algorithms exists: `class ROOT::Math::Minimizer`
- All minimizers in ROOT (Minuit, Minuit2, Fumili, GSL minimizers, simulated annealing, genetic) implement this interface
- Using the ROOT plug-in manager it is possible to change the implementation at run-time
- The interface can be used for fitting user defined likelihood or least-square functions
  - see ROOT [tutorial fit/NumericalMinimization.C](#) on how to use this interface

# Under the Print-Out: MIGRAD

- Purpose: find minimum

\*\*\*\*\*

\*\* 13 \*\*MIGRAD 1000 1

\*\*\*\*\*

(some output omitted)

MIGRAD MINIMIZATION HAS CONVERGED.

MIGRAD WILL VERIFY CONVERGENCE AND ERROR MATRIX  
COVARIANCE MATRIX CALCULATED SUCCESSFULLY

FCN=257.304 FROM MIGRAD STATUS=CONVERGED 31 CALLS 32 TOTAL  
EDM=2.36773e-06 STRATEGY= 1 ERROR MATRIX ACCURATE

EXT PARAMETER

NO. NAME

VALUE

ERROR

STEP

FIRST

SIZE

DERIVATIVE

1 mean

8.84225e-02

3.23862e-01

3.58344e-04

-2.24755e-02

2 sigma

3.20763e+00

2.39540e-01

2.78628e-04

-5.34724e-02

ERR DEF= 0.5

EXTERNAL ERROR MATRIX. NDIM= 25 NPAR 2 ERR DEF=0.5

1.049e-01 3.338e-04

3.338e-04 5.739e-02

PARAMETER CORRELATION COEFFICIENTS

NO.

GLOBAL

1

2

1

0.00430

1.000

0.004

2

0.00430

0.004

1.000

Progress information,  
watch for errors here

Parameter values and approximate  
errors reported by MINUIT

Error definition (in this case 0.5 for  
a likelihood fit)

# Under the Print-Out: MIGRAD

- Purpose: find minimum

\*\*\*\*\*

\*\* 13 \*\*MIGR

\*\*\*\*\*

(some output of

MIGRAD MINIMIZ

MIGRAD WILL VERI

COVARIANCE MATRIX CALCULATED SUCCESSFULLY

FCN=257.304

FROM MIGRAD

STATUS=CONVERGED

31 CALLS

32 TOTAL

EDM=2.36773e-06

STRATEGY= 1

ERROR MATRIX ACCURATE

EXT PARAMETER

STEP

FIRST

NO. NAME

VALUE

ERROR

SIZE

DERIVATIVE

1 mean

8.84225e-02

3.23862e-01

3.58344e-04

-2.24755e-02

2 sigma

3.20763e+00

2.39540e-01

2.78628e-04

-5.34724e-02

ERR DEF= 0.5

EXTERNAL ERROR MATRIX.

NDIM= 25

NPAR= 2

ERR DEF=0.5

1.049e-01 3.338e-04

3.338e-04 5.739e-02

PARAMETER CORRELATION COEFFICIENTS

NO. GLOBAL

1

2

1 0.00430 1.000 0.004

2 0.00430 0.004 1.000

Value of  $\chi^2$  or likelihood at minimum

(NB:  $\chi^2$  values are not divided by  $N_{d.o.f}$ )

Approximate  
Error matrix  
And covariance matrix



# Under the Print-Out: MIGRAD

- Purpose: find minimum

*Status:*  
Should be 'converged' but can be 'failed'

*Estimated Distance to Minimum*  
should be small  $O(10^{-6})$

*Error Matrix Quality*  
should be 'accurate', but can be  
'approximate' in case of trouble

\*\*\*\*\*

\*\* 13 \*\*MIGRAD 1000

\*\*\*\*\*

(some output omitted)

MIGRAD MINIMIZATION HAS CONVERGED

MIGRAD WILL VERIFY CONVERGENCE AND ESTIMATE ERROR MATRIX.

COVARIANCE MATRIX CALCULATED SUCCESSFULLY

FCN=257.304	FROM MIGRAD	STATUS=CONVERGED	31 CALLS	32 TOTAL
EDM=2.36773e-06		STRATEGY= 1	ERROR MATRIX ACCURATE	

EXT PARAMETER

NO.	NAME	VALUE	ERROR	STEP SIZE	FIRST DERIVATIVE
1	mean	8.84225e-02	3.23862e-01	3.58344e-04	-2.24755e-02
2	sigma	3.20763e+00	2.39540e-01	2.78628e-04	-5.34724e-02

ERR DEF= 0.5

EXTERNAL ERROR MATRIX. NDIM= 25 NPAR= 2 ERR DEF=0.5

1.049e-01 3.338e-04

3.338e-04 5.739e-02

PARAMETER CORRELATION COEFFICIENTS

NO.	GLOBAL	1	2
1	0.00430	1.000	0.004
2	0.00430	0.004	1.000

# Under the Print-Out: HESSE

- Purpose: calculate error matrix from  $\frac{d^2L}{dp^2}$

\*\*\*\*\*  
\*\* 18 \*\*HESSE 1000  
\*\*\*\*\*  
COVARIANCE MATRIX CALCULATED SUCCESSFULLY  
FCN=257.304 FROM HESSE STATUS=OK  
4e-06 STRAT  
TOTAL  
CURATE

Symmetric errors  
calculated from 2<sup>nd</sup>  
derivative of -ln(L) or  $\chi^2$

Error matrix  
(Covariance Matrix) calculated  
from  
$$V_{ij} = \left( \frac{d^2(-\ln L)}{dp_i dp_j} \right)^{-1}$$

EXTERNAL ERROR MATRIX  
1.049e-01 2.780e-04  
2.780e-04 5.739e-02

PARAMETER CORRELATION COEFFICIENTS  
NO. GLOBAL 1 2  
1 0.00358 1.000 0.004  
2 0.00358 0.004 1.000

Correlation matrix  $\rho_{ij}$   
calculated from  
$$V_{ij} = \sigma_i \sigma_j \rho_{ij}$$

Global correlation vector:  
correlation of each parameter with *all other*  
parameters

ERROR STEP SIZE INTERNAL VALUE  
3.23861e-01 7.16689e-05 8.84237e-03  
2.39539e-01 5.57256e-05 3.26535e-01  
ERR DEF= 0.5  
NDIM= 25 NPAR= 2 ERR DEF=0.5

# Under the Print-Out: MINOS

- Error analysis through  $\Delta n l$  contour finding

```
*****
**    23  **MINOS          1000
*****

FCN=257.304  FROM MINOS      STATUS=SUCCESSFUL      52  CALLS      94  TOTAL
                EDM=2.36534e-06      STRATEGY= 1      ERROR MATRIX ACCURATE

EXT  PARAMETER
NO.   NAME      VALUE
  1  mean      8.84225e-02
  2  sigma     3.20763e+00

PARABOLIC
ERROR
3.23861e-01
2.39539e-01

MINOS ERRORS
NEGATIVE    POSITIVE
-3.24688e-01  3.25391e-01
-2.23321e-01  2.58893e-01

ERP DEF= 0.5
```

**Symmetric error**  
(repeated result  
from HESSE)

**MINOS error**  
Can be asymmetric  
(in this example the 'sigma' error  
is slightly asymmetric)

# 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

$$T_0(x) = 1$$

$$T_1(x) = x$$

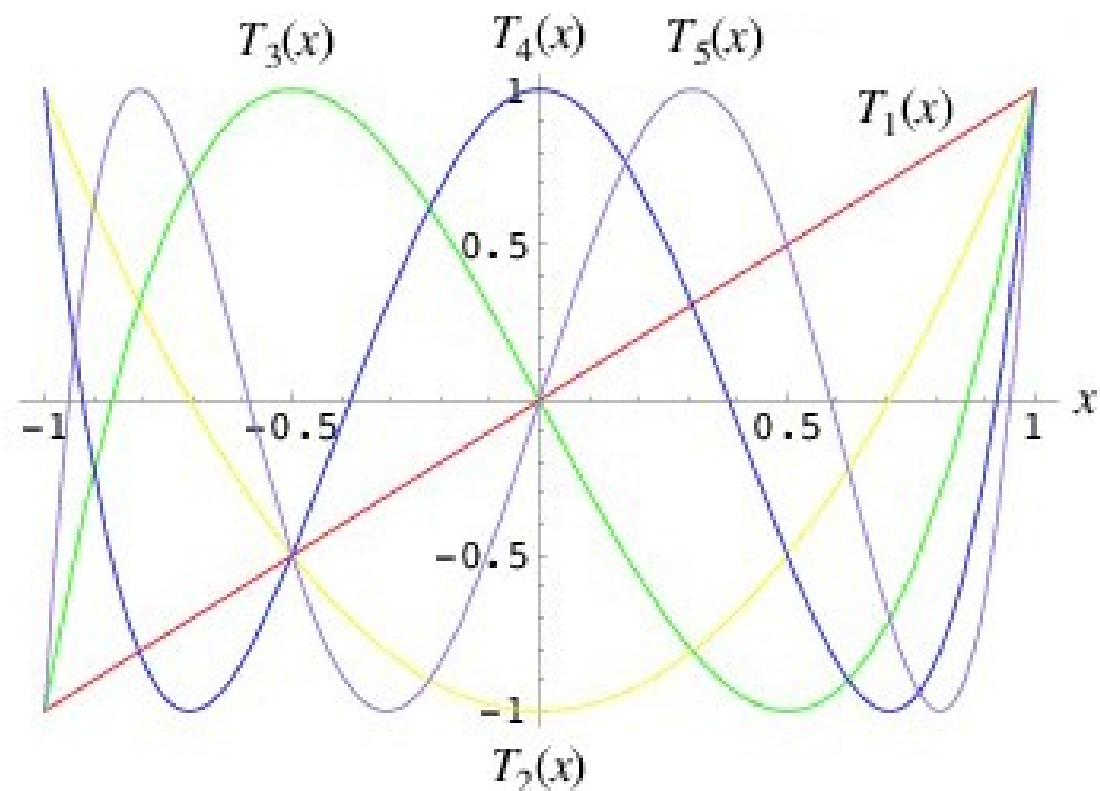
$$T_2(x) = 2x^2 - 1$$

$$T_3(x) = 4x^3 - 3x$$

$$T_4(x) = 8x^4 - 8x^2 + 1$$

$$T_5(x) = 16x^5 - 20x^3 + 5x$$

$$T_6(x) = 32x^6 - 48x^4 + 18x^2 - 1.$$



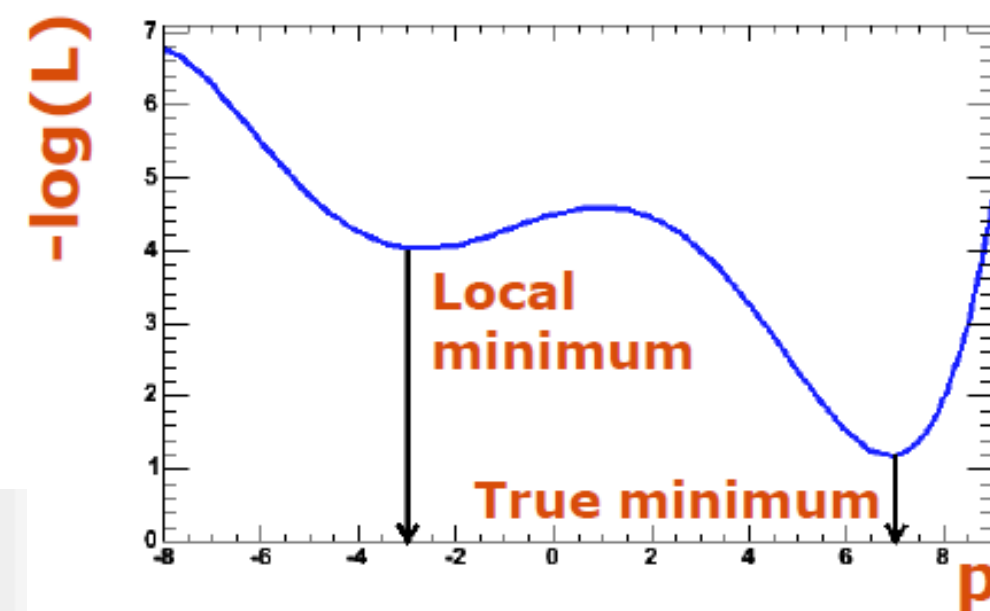
# To Get the Truth from fitting

- 利用 **MINUIT** 函数 **MIGRAD** 找  $-\log L$  的最小值
- 利用 **MINUIT** 函数 **HESSE** 计算参数的误差
- 还可利用 **MINUIT** 函数 **MINOS** 做更好的误差估计

❑ **HESSE** 相关系数矩阵是很好的调查起点

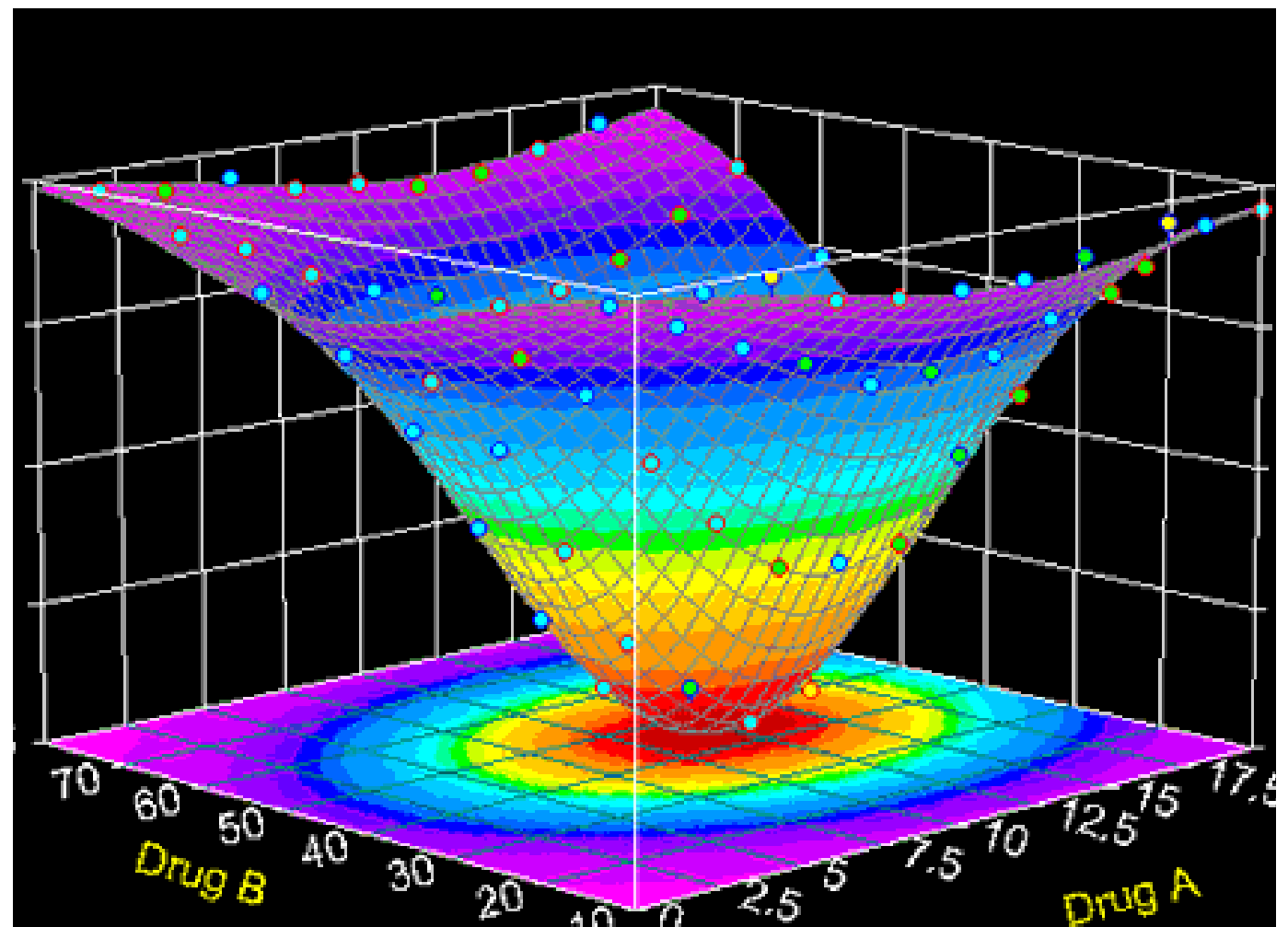
PARAMETER	CORRELATION COEFFICIENTS		
NO.	GLOBAL	1	2
1	0.99835	1.000	0.998
2	0.99835	0.998	1.000

表明两个参数几乎 **100%** 关联，求最小值过程无法建立。



# To Get the Right Answer from Minuit

- Limits on variable parameters should be used only when needed in order to prevent from taking on unphysical values
- When a satisfactory minimum has been found using limits, the limits should then be removed if possible, in order to perform or re-perform the error analysis without limits
- Minuit offers several minimization algorithms: **MIGRAD**, **HESSE**, **MINOS** etc al. The MIGRAD algorithm is in general the best.
- If parameter limits are needed, one should be aware of some techniques to alleviate problems caused by limits.



# To Get the Right Answer from Minuit

## □ Getting the Right Minimum with Limits:

If MIGRAD converges normally to a point where no parameter is near one of its limits, the existence of limits has probably not prevented Minuit from finding the right minimum. On the other hand, if one or more parameters is near its limit at the minimum, this may be because the true minimum is indeed at a limit, or it may be because the minimized has become "blocked" at a limit.

## □ Getting the Right Parameter Errors with Limits

In the best case, where the minimum is far from any limits, Minuit will correctly transform the error matrix, and the parameter errors it reports should be accurate. In other cases (which should be more common, since otherwise you would not need limits), the very meaning of parameter errors becomes problematic.

# Reliability of Minuit Error Estimates

- When **Minuit** print-out gives the indications of error values with **CURRENT GUESS ERROR** or **APPROXIMATE ERROR**, it means that the errors have been calculated but they may not be accurate.
- Some signs that the errors may not be accurate:
  - ▣ Warning messages produced during the minimization or error analysis
  - ▣ Failure to find new minimum
  - ▣ Value of EDM too big (Estimated Distance to Minimum)( $>10^{-3}$ )
  - ▣ Correlation coefficients exactly equal to zero, unless some parameters are known to be uncorrelated with the others
  - ▣ Correlation coefficients very close to one(greater than 0.99).
  - ▣ Parameter at limit.

# Covariance Matrix

- Theoretically, the covariance matrix for a "physical" function must be positive-definite at the minimum. Therefore, if **MIGRAD** reports a non-positive-definite covariance matrix, this may be a sign of the following:
  - ▣ **A Non-physical Region:** leave such a region.
  - ▣ **An Underdetermined Problem:** reformulate the function
  - ▣ **Numerical Inaccuracies:** excessive round off errors or not enough precision, more likely if the number of free parameters is very large, or if the parameters are badly scaled and correlations are large.
- **An ill-posed Problem:**
  - ▣ Excessive numerical round off – especially for exponential and factorial functions which get big very quickly and lose accuracy.
  - ▣ Starting too far from the solution - the function may have unphysical local minima

# Suggestions on Using Minuit

- Read Minuit reference manual *minuit.ps*: In addition to the obligatory technicalities, read chapters entitled: “Minuit Basic Concepts” ; “How to get the right answer from Minuit”; “Interpretation of the errors on Minuit parameters” .
- Provide initial values reasonably close to the true minimum: save time; the mean/median ...
- To keep off from unphysical region: limit; FCN definition; warning message;
- Minuit strategy level: 0;1;2
- Numerical precision: SET EPS;
- In cases where Minuit cannot locate minimum: try several precedent experimental fits where subset of the parameters are fixed;

# FUMILI Minimization Package

- **TFumili** is an optimized method for chi-square and log likelihood minimizations.

- The minimum condition is: 
$$\frac{\partial \chi^2}{\partial \theta_i} = \sum_{j=1}^n \frac{1}{\sigma_j^2} \cdot \frac{\partial f_j}{\partial \theta_i} [f_j(\vec{x}_j, \vec{\theta}) - F_j] = 0, \quad i = 1 \dots m$$

Expand the left part of the above equation:

$$\left( \frac{\partial \chi^2}{\partial \theta_i} \right)_{\theta=\vec{\theta}^0} + \sum_k \left( \frac{\partial^2 \chi^2}{\partial \theta_i \partial \theta_k} \right)_{\theta=\vec{\theta}^0} \cdot (\theta_k - \theta_k^0) = 0$$

here  $\vec{\theta}_0$  is some initial value of parameters.

$$\frac{\partial^2 \chi^2}{\partial \theta_i \partial \theta_k} = \sum_{j=1}^n \frac{1}{\sigma_j^2} \cdot \frac{\partial f_j}{\partial \theta_i} \frac{\partial f_j}{\partial \theta_k} + \sum_{j=1}^n \frac{(f_j - F_j)}{\sigma_j^2} \cdot \frac{\partial^2 f_j}{\partial \theta_i \partial \theta_k}$$

- In FUMILI algorithm, the last term is discarded:

$$\frac{\partial^2 \chi^2}{\partial \theta_i \partial \theta_k} \approx z_{ik} = \sum_{j=1}^n \frac{1}{\sigma_j^2} \frac{\partial f_j}{\partial \theta_i} \frac{\partial f_j}{\partial \theta_k}$$



# FUMILI Minimization Package

- Then the equations for parameter increments are:

$$\left( \frac{\partial \chi^2}{\partial \theta_i} \right)_{\vec{\theta}=\vec{\theta}^0} + \sum_k z_{ik} \cdot (\theta_k - \theta_k^0) = 0, \quad i = 1 \dots m$$

For an initial value of parameter  $\vec{\theta}^0$  a parallelepiped  $P_0$  is built with the center at  $\vec{\theta}^0$  and axes parallel to coordinate axes  $\theta_i$ . The lengths of parallelepiped sides along  $i$ -axis is  $2b_i$ , where  $b_i$  is such a value that the functions  $f_j(\vec{\theta})$  are quasi-linear all over the parallelepiped. FUMILI takes into account simple linear inequalities in the form:  $\theta_i^{\min} \leq \theta_i \leq \theta_i^{\max}$ .

- Very similar step formulae are used in FUMILI for negative logarithm of the likelihood function with the same idea --linearization of functional argument.

*ref: FUMILI\_NIMA400.pdf (Nucl.Instrum.Meth.A440:431-437,2000)*

# Generating Pseudo Data

- Random number generation in ROOT is done using the TRandom classes
    - Three pseudo-random number generator exists, TRandom1, TRandom2 and TRandom3. TRandom is the base class.
      - TRandom3 (a Mersenne-Twister generator) is used by default (it has a very long period,  $\sim 10^{6000}$  and it is very fast).
      - Random numbers can be generated using the global static variable gRandom
- ```
root [0] gRandom->Rndm()  
(Double_t) 9.99741748906672001e-01
```
- TRandom::Rndm() generates uniform number in the [0,1] range.
  - Seeding is controlled using TRandom::SetSeed(seed).
  - When using seed = 0, independent random streams can be generated (the seed is based on a UUID number).

Universally Unique Identifier, 简称UUID,  
UUID是由一组32位数的16进制数字所构成

# Random Number Distributions

- The class `TRandom` provides methods to generate numbers according to some pre-defined distributions

| Distributions                                                                              | Description                                                                                           |
|--------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|
| <code>Double_t Uniform(Double_t x1, Double_t x2)</code>                                    | Uniform random numbers between <code>x1</code> , <code>x2</code>                                      |
| <code>Double_t Gaus(Double_t mu, Double_t sigma)</code>                                    | Gaussian random numbers.<br>Default values: <code>mu=0</code> , <code>sigma=1</code>                  |
| <code>Double_t Exp(Double_t tau)</code>                                                    | Exponential random numbers with mean <code>tau</code> .                                               |
| <code>Double_t Landau(Double_t mean, Double_t sigma)</code>                                | Landau distributed random numbers.<br>Default values: <code>mean=0</code> , <code>sigma=1</code>      |
| <code>Double_t BreitWigner(Double_t mean,<br/>Double_t gamma)</code>                       | Breit-Wigner distributed random numbers.<br>Default values <code>mean=0</code> , <code>gamma=1</code> |
| <code>Int_t Poisson(Double_t mean)</code><br><code>Double_t PoissonD(Double_t mean)</code> | Poisson random numbers                                                                                |
| <code>Int_t Binomial(Int_t ntot, Double_t prob)</code>                                     | Binomial Random numbers                                                                               |
| <code>Circle(Double_t &amp;x, Double_t &amp;y, Double_t r)</code>                          | Generate a random 2D point <code>(x, y)</code> in a circle of radius <code>r</code>                   |
| <code>Sphere(Double_t &amp;x, Double_t &amp;y,<br/>Double_t &amp;z, Double_t r)</code>     | Generate a random 3D point <code>(x, y, z)</code> in a sphere of radius <code>r</code>                |
| <code>Rannor(Double_t &amp;a, Double_t &amp;b)</code>                                      | Generate a pair of Gaussian random numbers with <code>mu=0</code> and <code>sigma=1</code>            |

# Random Number Distributions (2)

- Random numbers can be generated according to what-ever distribution using accept-rejection techniques (often not very efficient) or by using the inverse of the cumulative (integral) distribution
- RooFit use accept-rejection techniques (Hit or miss method).

```
RooGaussian gauss("gauss","gaussian PDF",  
                  x, mu, sigma);  
RooDataSet* data = gauss.generate(x, 10000);
```

- ROOT has the method `TF1::GetRandom()`, which uses this technique to generate random numbers from a generic function object

```
TF1 f(...);  
double x = f.GetRandom();  
TH1D histo(...);  
histo.FillRandom(f, 1000);
```

Adopted technique: binned cumulative inversion

**Caveat:** approximations may depend on internal function binning.

– Can change it using: **f.Npx(5000);**

# Summary

- We have learned:
  - the concept of fitting,
  - how to fit a histogram in ROOT.
- We have also learned:
  - how to generate random numbers and distributions which can be used to test and validate the fitting procedure.
- We will see also how fitting can be facilitate by using a tool like RooFit.

# Dig \$ROOTSYS/tutorials/fit

- Run the example macros
- Try to understand the codes of the example macros:  
references: <http://root.cern.ch/drupal/content/reference-guide>  
class header descriptions
- “*grep*” those macros to find what you need