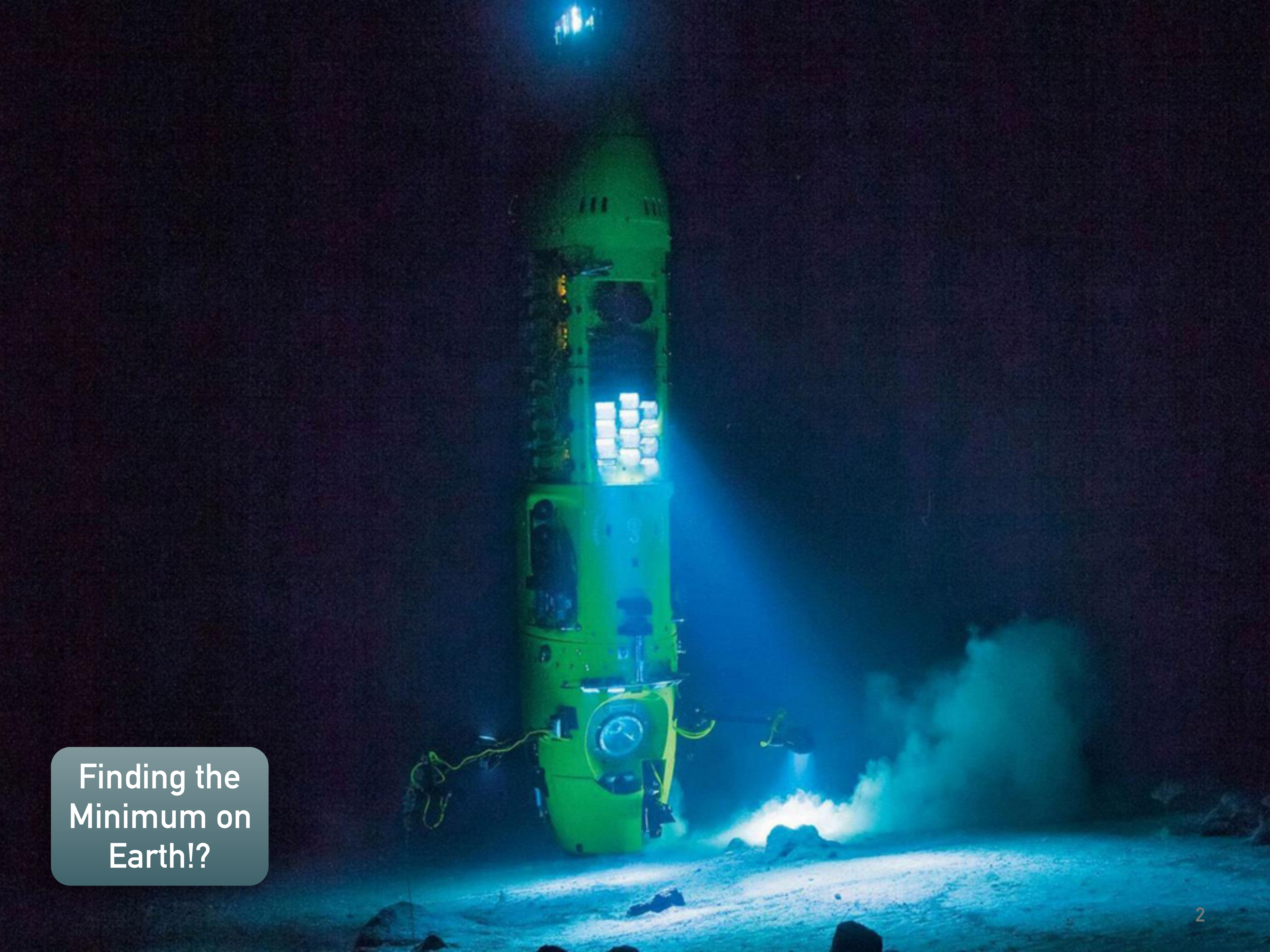
INTER-LECTURE: FITTING WITH MINUIT & ROOFIT

STATISTICAL ANALYSIS
IN EXPERIMENTAL PARTICLE PHYSICS

Kai-Feng Chen
National Taiwan University



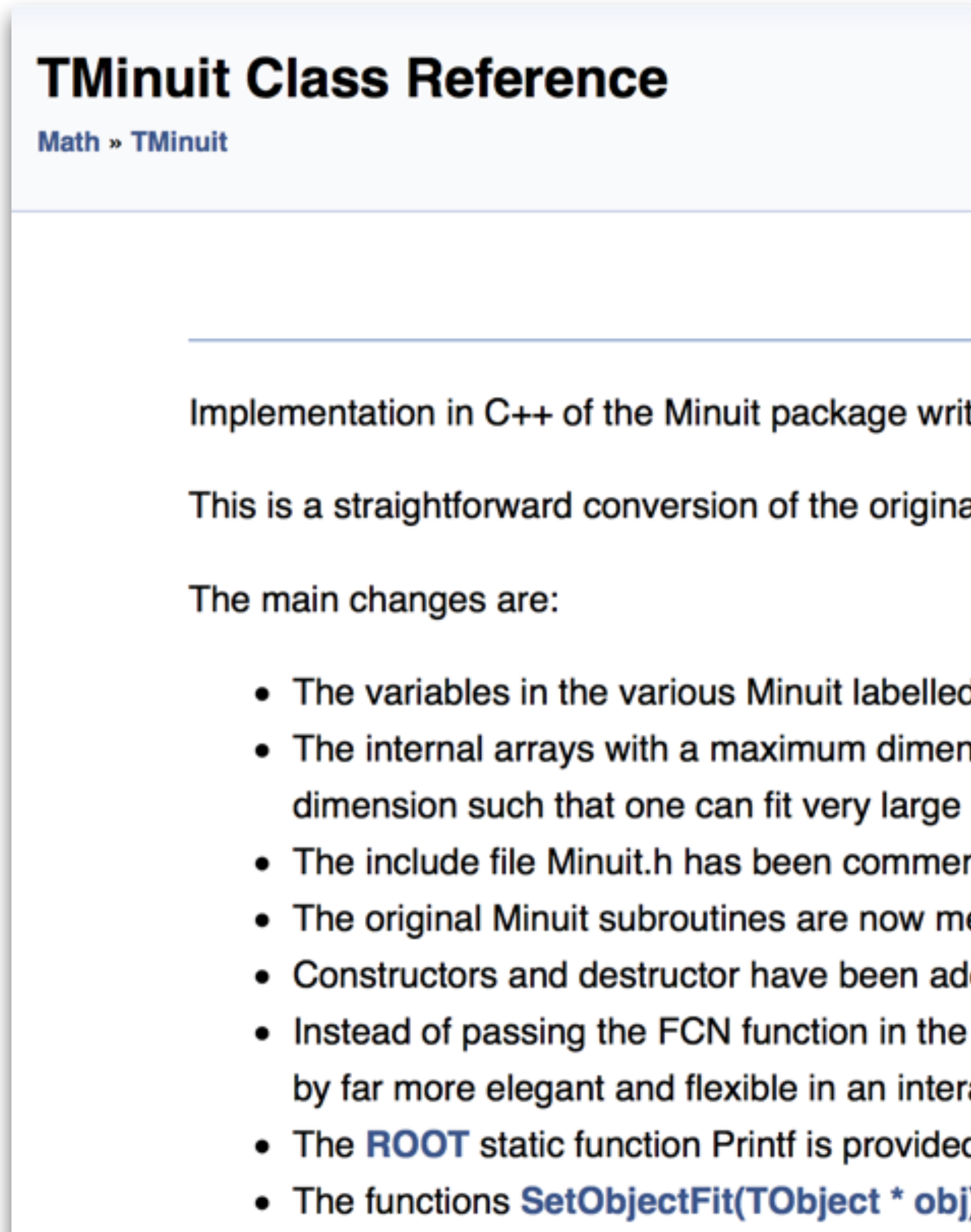
Finding the Minimum on Earth!?

FINDING MINIMUM WITH MINUIT

- In this inter-lecture, we are going to discuss the very well-known package: **MINUIT**.
- Minuit is conceived as a tool to find the minimum value of a multi-parameter function and analyze the shape of the function around the minimum.
- The principal application is foreseen for statistical analysis, working on chi-square or log-likelihood functions, to compute the best-fit parameter values and uncertainties, including correlations between the parameters.
- It is especially suited to handle difficult problems, including those which may require guidance in order to find the correct solution.
- It was a part of CERN library (written in fortran), but it has been merged/migrate into ROOT and its C++ successors (Minuit++, Minuit2, etc).

FINDING MINIMUM WITH MINUIT (CONT.)

- Basically one can still use the original implement in the nice-old CERNLIB and either write a fortran code or any working C/C++ wrappers to carried out the job.
- Here we will adopt a very easy way to use it (probably the easiest one?) –– Call it with **ROOT::TMinuit** class.
- So you only need a working ROOT.



TMinuit Class Reference

Math » TMinuit

Implementation in C++ of the Minuit package writ

This is a straightforward conversion of the origina

The main changes are:

- The variables in the various Minuit labelled
- The internal arrays with a maximum dimen dimension such that one can fit very large
- The include file Minuit.h has been commen
- The original Minuit subroutines are now m
- Constructors and destructor have been ad
- Instead of passing the FCN function in the by far more elegant and flexible in an intera
- The ROOT static function Printf is provided
- The functions SetObjectFit(TObject * obj

STRUCTURE OF A MINUIT PROGRAM

➤ Let's perform a 2D minimum finding with the TMinuit class:

example_01.cc

```
void fcn(int &npar, double *gin, double &f, double *par, int iflag)
{
    double x = par[0], y = par[1];
    f = pow(x-2.,2)+pow(y-3.,2);
}                                                  core "fcn" function

void example_01()
{                                        ↶ initial a TMinuit object
    TMinuit *gMinuit = new TMinuit(2);     with 2 parameters
    gMinuit->SetFCN(fcn);
    gMinuit->DefineParameter(0, "x", 8. , 1., 0., 0.);↶ definition of variables
    gMinuit->DefineParameter(1, "y", 6. , 1., 0., 0.);

    gMinuit->Command("MIGRAD"); ↶ execute Minuit commands
    gMinuit->Command("MIGRAD");

    double x,y,xerr,yerr;
    gMinuit->GetParameter(0,x,xerr);
    gMinuit->GetParameter(1,y,yerr);

    printf("x: %+.7f +- %.7f\n",x,xerr);
    printf("y: %+.7f +- %.7f\n",y,yerr);
}
```

THE CORE FCN FUNCTION

- The most important function is the **"FCN"**. The user of Minuit must always supply a routine which calculates the function value to be minimized or analyzed.
- The structure:

```
void fcn(int &npar, double *gin, double &f, double *par, int iflag)
{
    double x = par[0], y = par[1];
    f = pow(x-2.,2)+pow(y-3.,2);
}
```

↶ the returned value must be set to the reference "f"

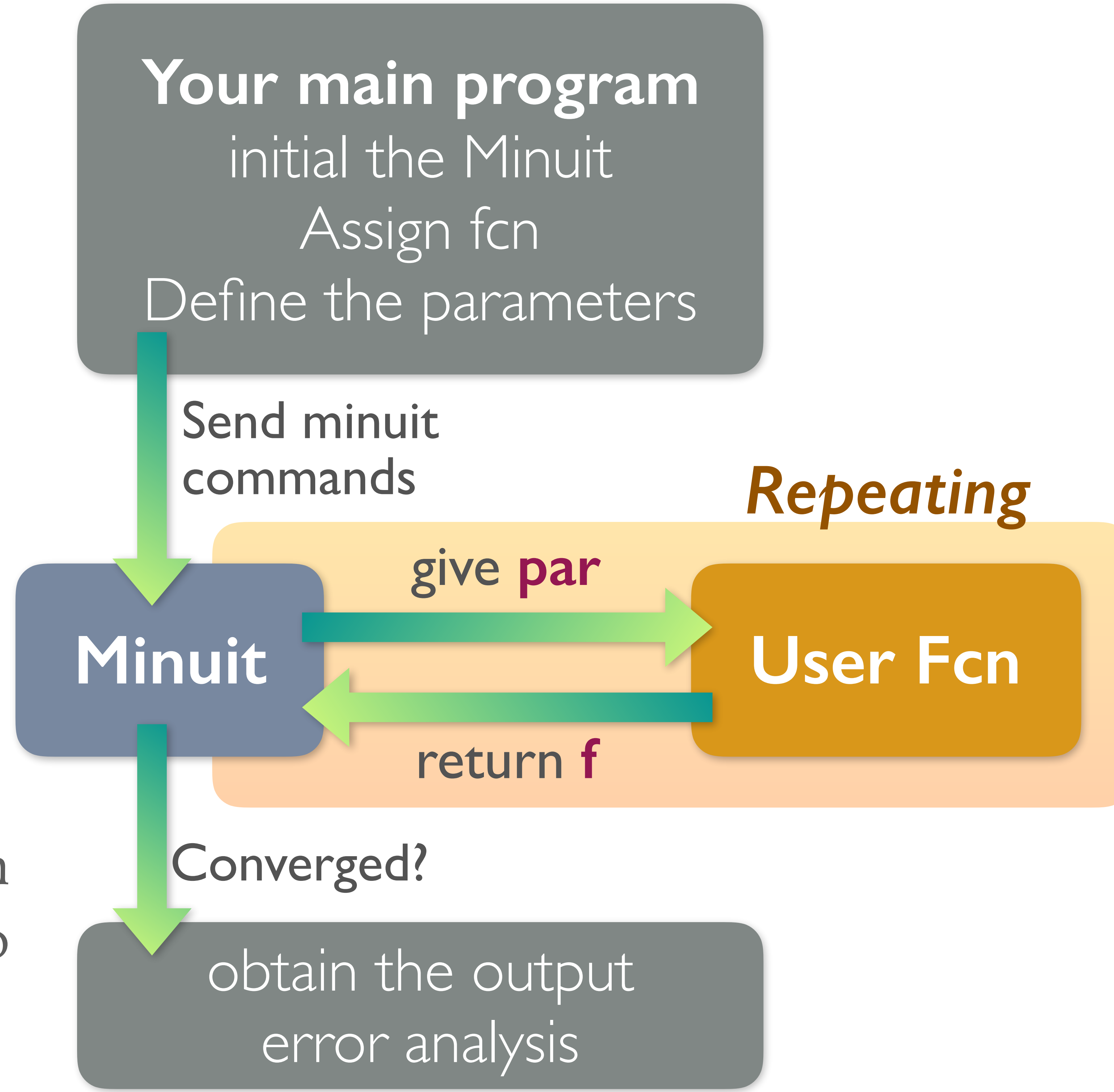
f & par are required; you can ignore others in a simple problem solving program.

npar – number of currently variable parameters.
gin – the (optional) vector of first derivatives.
f – the calculated function value.
par – vector of (constant and variable) parameters.
iflag – indicates the stage of minimization

THE WORKFLOW

Here are how a Minuit program runs:

- Your main program has to initialize the TMinuit class and provide your core fcn function.
- Parameters have to be defined, either floated or fixed.
- Send the corresponding commands to Minuit, which will call your fcn function to obtain the function values.



EXECUTE IT

- Terminal output (as a classical screenshot from Minuit):

```
Processing example_01.cc...
 PARAMETER DEFINITIONS:
    NO.    NAME         VALUE      STEP SIZE      LIMITS
     1 x           8.00000e+00  1.00000e+00     no limits
     2 y           6.00000e+00  1.00000e+00     no limits
 **********
 **    2 **MIGRAD
 **********
 . . .
 FCN=0 FROM MIGRAD    STATUS=CONVERGED       9 CALLS          33 TOTAL
                     EDM=0    STRATEGY= 1      ERROR MATRIX ACCURATE
  EXT PARAMETER                                  STEP         FIRST
  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
   1  x            2.00000e+00   1.00000e+00  -1.98483e-10   0.00000e+00
   2  y            3.00000e+00   1.00000e+00  -9.87050e-11   0.00000e+00
 EXTERNAL ERROR MATRIX.    NDIM=  25    NPAR=  2    ERR DEF=1
  1.000e+00 -1.110e-16
 -1.110e-16  1.000e+00
 PARAMETER  CORRELATION COEFFICIENTS
       NO.  GLOBAL      1      2
        1  0.00000   1.000 -0.000
        2  0.00000  -0.000  1.000
x: +2.0000000 +- 1.0000000
y: +3.0000000 +- 1.0000000
```

↶ final returned fcn value

↶ minimization status

Meaning of the errors: to be discussed in the next lecture!

↶ perfect solution!

Run the code under ROOT

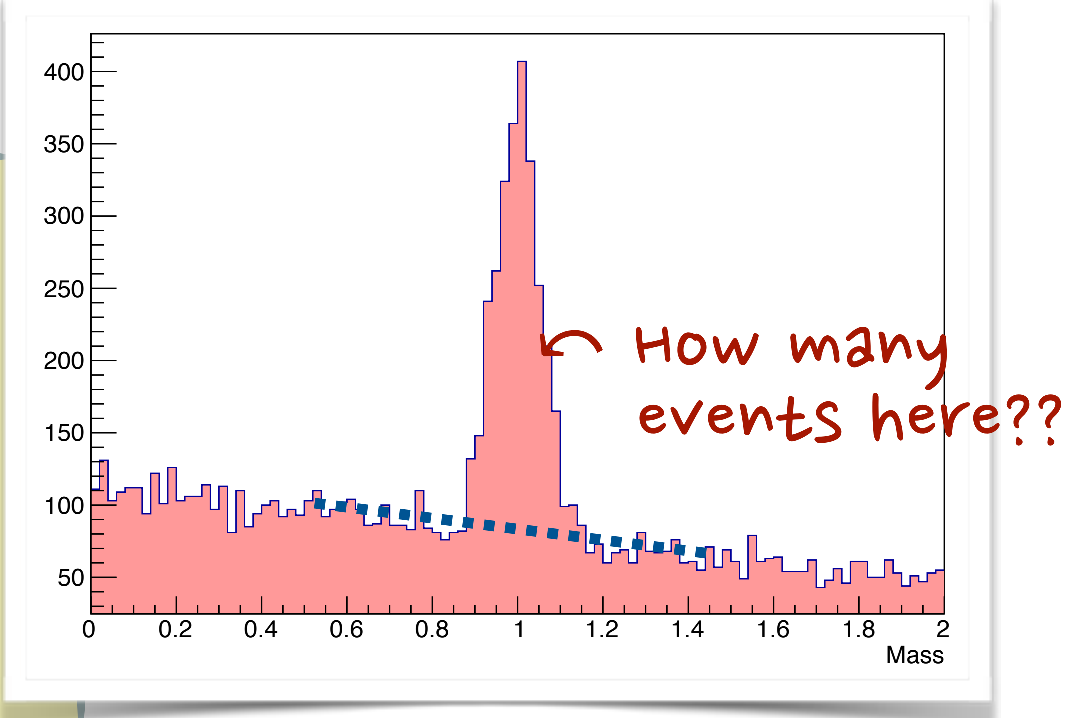
A TYPICAL FIT EXAMPLE

- As a typical using Minuit, is to perform a fit on a given distribution.
- One of the most typical examples is to extract the signal out of a mixture of signal and background. e.g., you may obtain a set of data looks like below.
- Please download the test pseudo data file from http://hep1.phys.ntu.edu.tw/~kfjack/lecture/hepstat/in3/example_data.root

You will find a histogram and ntuple, both of them contain the same data!

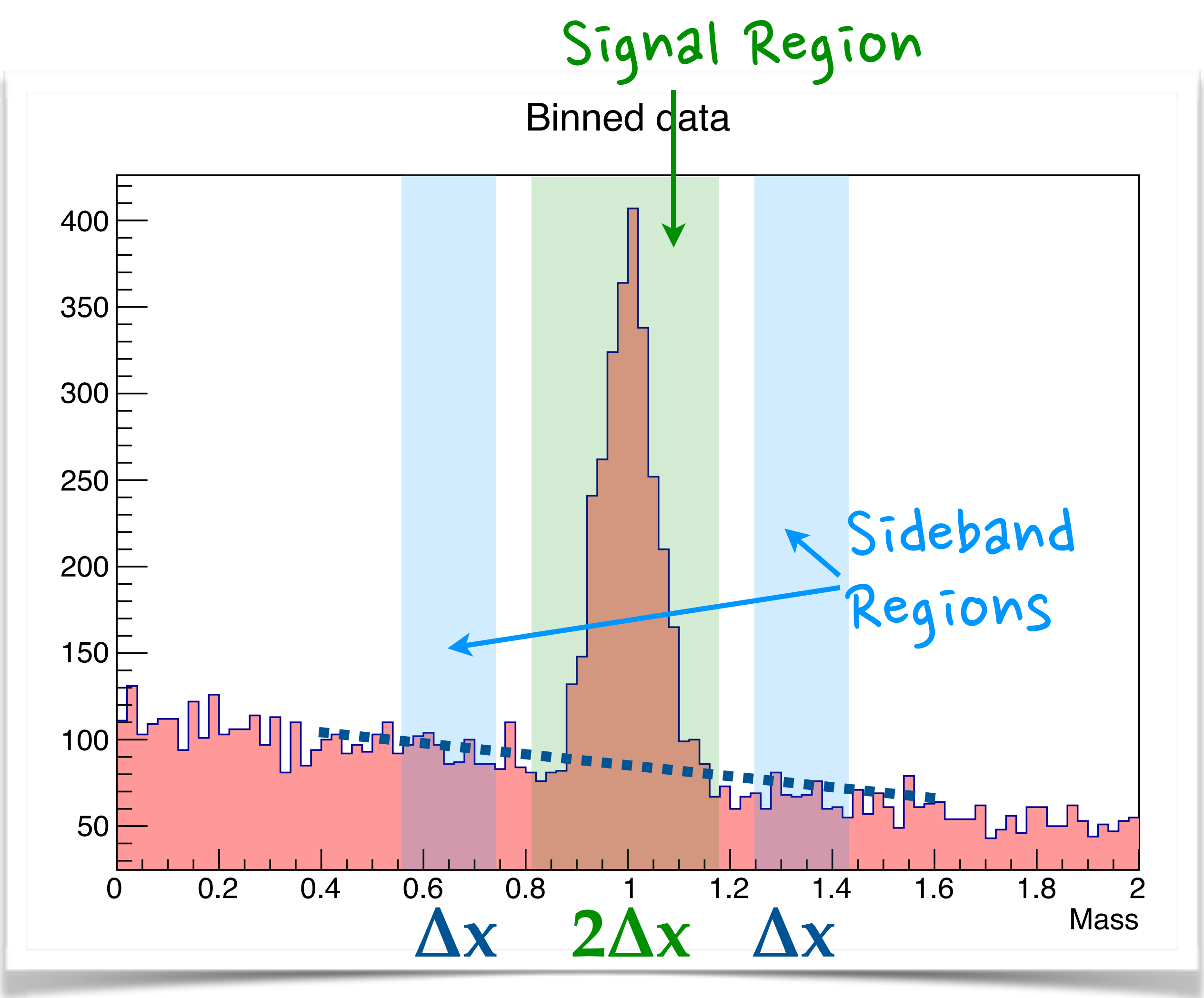
example_02.cc

```
{
    TFile *fin = new TFile("example_data.root");
    TH1D *hist = (TH1D *)fin->Get("hist");

    hist->SetFillColor(kRed-9);
    hist->SetStats(false);
    hist->GetXaxis()->SetTitle("Mass");
    hist->Draw();
}
```



SIDEBAND SUBTRACTION

- Usually adopting a "sideband subtraction" is the simplest way to extract the yield of signal under the "peaking structure".
- However one still need to assume that "background is (*nearly*) flat" in the calculation of background.



As far as the width of signal region is the same as the sideband regions, the # of signal can be extracted by:

$$S = N_{\text{signal region}} - N_{\text{sideband}}$$

uncertainty (since the yields are Poisson distributed!):

$$\Delta S = \sqrt{\Delta N^2_{\text{signal region}} + \Delta N^2_{\text{sideband}}} = \sqrt{N_{\text{signal region}} + N_{\text{sideband}}}$$

SIDEBAND SUBTRACTION (CONT.)

➤ If you know the **mean & width of the signal**, this is quick and easy!

example_03.cc

```
{
    TFile *fin = new TFile("example_data.root");
    TNtupleD *nt = (TNtupleD *)fin->Get("nt");

    const double MEAN = 1.0;
    const double SIGMA = 0.05;

    int count_sigregion = 0, count_sideband = 0;
    for(int evt=0; evt<nt->GetEntries(); evt++) {
        nt->GetEntry(evt);
        double mass = nt->GetArgs()[0];

        if (fabs(mass-MEAN)<SIGMA*3.) count_sigregion++;
        if (fabs(mass-MEAN)>SIGMA*3.5 &&
            fabs(mass-MEAN)<SIGMA*6.5) count_sideband++;
    }

    double S = count_sigregion - count_sideband;
    double dS = sqrt(count_sigregion+count_sideband);
    printf("N(sig) = %.1f +- %.1f\n",S,dS);
}
```

Terminal output

```
Processing example_03.cc...
N(sig) = 2035.0 +- 66.3
```

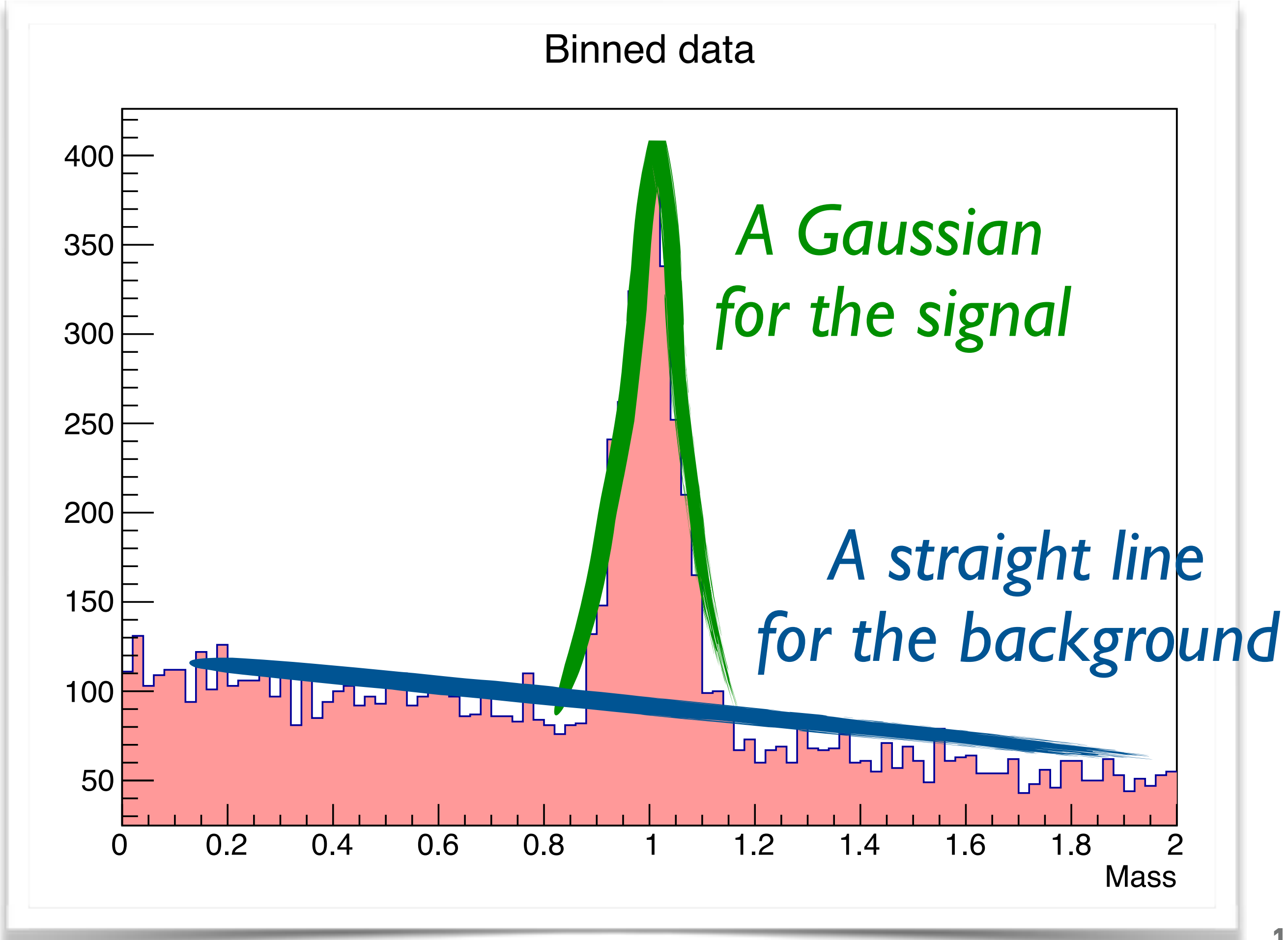
Signal region: within 3 sigma

Sideband region: 3.5~6.5 sigma

ESTIMATION WITH FITS

- Surely we are going to introduce something more sophisticated than a simple subtraction of sideband candidates.
- Probably this is the most straightforward modeling in many cases?

From the following slides we are going to discuss several parameter methods and its limitation. A more detailed discussions (mathematics!) will be introduced in the next lecture!

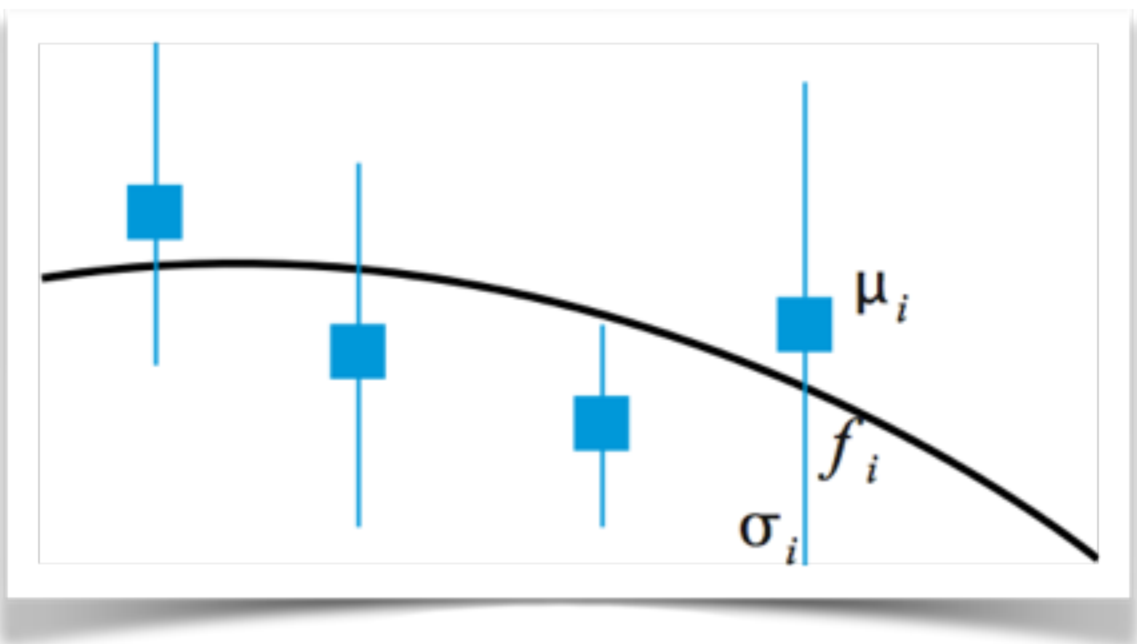


ESTIMATION WITH A LEAST-SQUARE FIT

- The result of the best fits can be obtained by minimizing a χ^2 value for ***N*** independent measurements:

$$\chi^2 = \sum_i^N \frac{(f_i - \mu_i)^2}{\sigma_i^2}$$

f_i: expected value of the model
μ_i: i^{th} measurement
σ_i: uncertainty of i^{th} measurement



Keeping updating the parameters (α,β,γ,....) until the **best (smallest)** χ^2 is reached.

$$f_i = f(x_i; \alpha, \beta, \gamma, \ldots)$$

Data should be binned in this case!

FITTING UTILITIES IN ROOT

- The embedded fitting routine in **histogram class** is one of the trivial methods to perform a χ^2 fit:

example_04.cc

```
{
    TFile *fin = new TFile("example_data.root");
    TH1D *hist = (TH1D *)fin->Get("hist");

    TF1 *f1 = new TF1("f1","[0]+[1]*x+[2]*gaus(2)");
    f1->SetParameters(100.,-30.,20.,1.,0.05);
    hist->Fit("f1");
}
```

$$f(x) = c + b \cdot x + a \cdot G(x; \mu, \sigma)$$

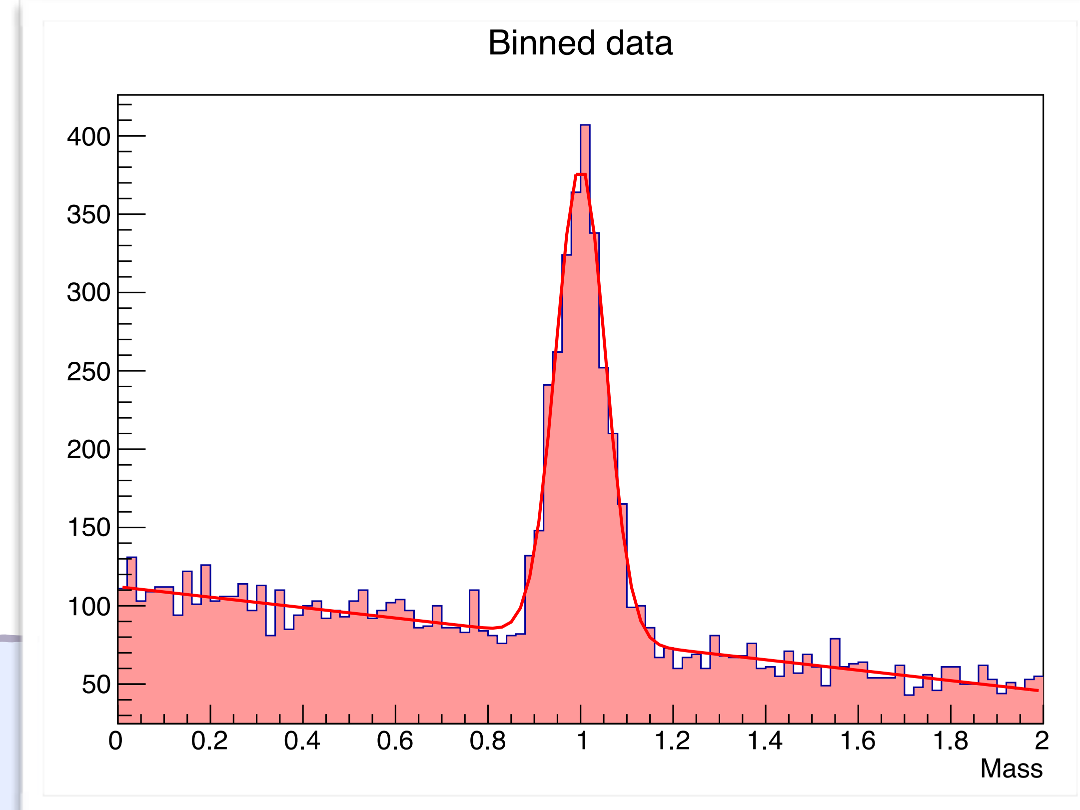
$$G(x; \mu, \sigma) = \exp(-\frac{(x-\mu)^2}{2\sigma^2})$$ ***(a non-normalize Gaussian)***

Given the initial values: $c = 100., b = -30, a = 20., \mu = 1.0, \sigma = 0.05$

As far as we have a rough guess for the initial values, the fitter will help to find the best values according to the χ2 value.

FITTING UTILITIES IN ROOT (CONT.)

- You may find a familiar screen printout from Minuit:




```
% root -l example_04.cc
root [0]
Processing example_04.cc...
FCN=93.6468 FROM MIGRAD    STATUS=CONVERGED      148 CALLS         149 TOTAL
EDM=1.10452e-09    STRATEGY= 1      ERROR MATRIX ACCURATE
  EXT PARAMETER                                STEP         FIRST
  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
   1  p0           1.12163e+02   2.00591e+00   4.24707e-03  -1.73048e-05
   2  p1          -3.33062e+01   1.52152e+00   3.27049e-03  -1.58848e-05
   3  p2           1.73735e+01   2.87792e-01   1.10339e-03   7.43857e-05
   4  p3           1.00031e+00   1.70386e-03   8.08886e-06  -1.05440e-03
   5  p4           5.34156e-02   1.51018e-03   5.68064e-06  -1.97576e-02
root [1]
```

COMMENTS

- How about the $\chi 2$ value?
  A: The FCN value is exactly the final χ^2.

```
FCN=93.6468
```

- Whats are the measurements and the associated uncertainties?
  A: the height of the bins in the histogram is the measurement; the error is just square-root of the height. This is the Poisson error as introduced in the previous lecture.

- N(degree of freedom) in the fit?
  A: **N(d.o.f.) = N(bins) – N(free parameters) = 100 – 5 = 95**.

- What's the confidence level?
  A: Use the ROOT TMath::Prob() command, it's ~52%.

```
root [0] TMath::Prob(93.6468,95)
(double) 0.520016
```

BINNED LIKELIHOOD FIT?

- Another commonly used estimator (*actually a better one!*) is the binned likelihood method. This can be performed easily by supply the proper commend to TH1:Fit().
- We will come back to this in detail in the next lecture.

partial example_04a.cc

```
hist->Fit("f1","L");
```

You may find the results are consistent but slightly different.

```
FCN=47.0652 FROM MIGRAD    STATUS=CONVERGED     148 CALLS     149 TOTAL
EDM=2.44648e-08    STRATEGY= 1      ERROR MATRIX ACCURATE
 EXT PARAMETER                                   STEP         FIRST
 NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
  1  p0           1.13095e+02   1.99405e+00   4.27946e-03   7.50878e-05
  2  p1          -3.33650e+01   1.50439e+00   3.27655e-03   2.51270e-05
  3  p2           1.73676e+01   2.88202e-01   1.10629e-03   1.48242e-04
  4  p3           1.00018e+00   1.70197e-03   8.10039e-06   3.92049e-02
  5  p4           5.36192e-02   1.50342e-03   5.67091e-06   7.95277e-02
```

COMMENTS (II)

- This is very quick and easy if we just want to do some simple fits (*only 3 extra lines as in the example code!*).
- However, this fit has no full control of fitting commands, no control of error definitions, and no control of detailed parameter setup.
- It's extremely hard to guess initial values, especially in this kind of naïve coding. Hence it is hard to do a more complicated fit, or with complex fitting functions/models.
- In any case this "root fit" works very well very similar cases.
- Hard to extract the results in your own favored way. (*e.g. what's the area of the Gaussian signal? we have do some integration afterwards…*)
    - ➡ **Let's move on and prepare a fitter based on Minuit directly, as a good practice!**

FIT WITH MINUIT

➤ The “modeling” part of the example code:

partial example_05.cc

```
TH1D *hist = 0;

double model(double x, double *par)
{
    double mu    = par[3];
    double sigma = par[4];
    double norm  = 1./sqrt(2.*TMath::Pi())/sigma;
    double G     = norm*exp(-0.5 *pow((x-mu)/sigma,2));
    double BinWidth = hist->GetBinWidth(1);

    return par[0] + par[1]*x + par[2] * BinWidth * G;
}

void fcn(int &npar, double *gin, double &f, double *par, int iflag)
{
    f = 0.;
    for(int i=1;i<=hist->GetNbinsX();i++) {
        double x       = hist->GetBinCenter(i);
        double measure = hist->GetBinContent(i);
        double error   = sqrt(measure);
        double func    = model(x,par);
        double delta   = (func - measure)/error;

        f += delta*delta;
    }
}
```

Gaussian function (normalized!)

calculate $f = \chi^2$

partial example_05.cc

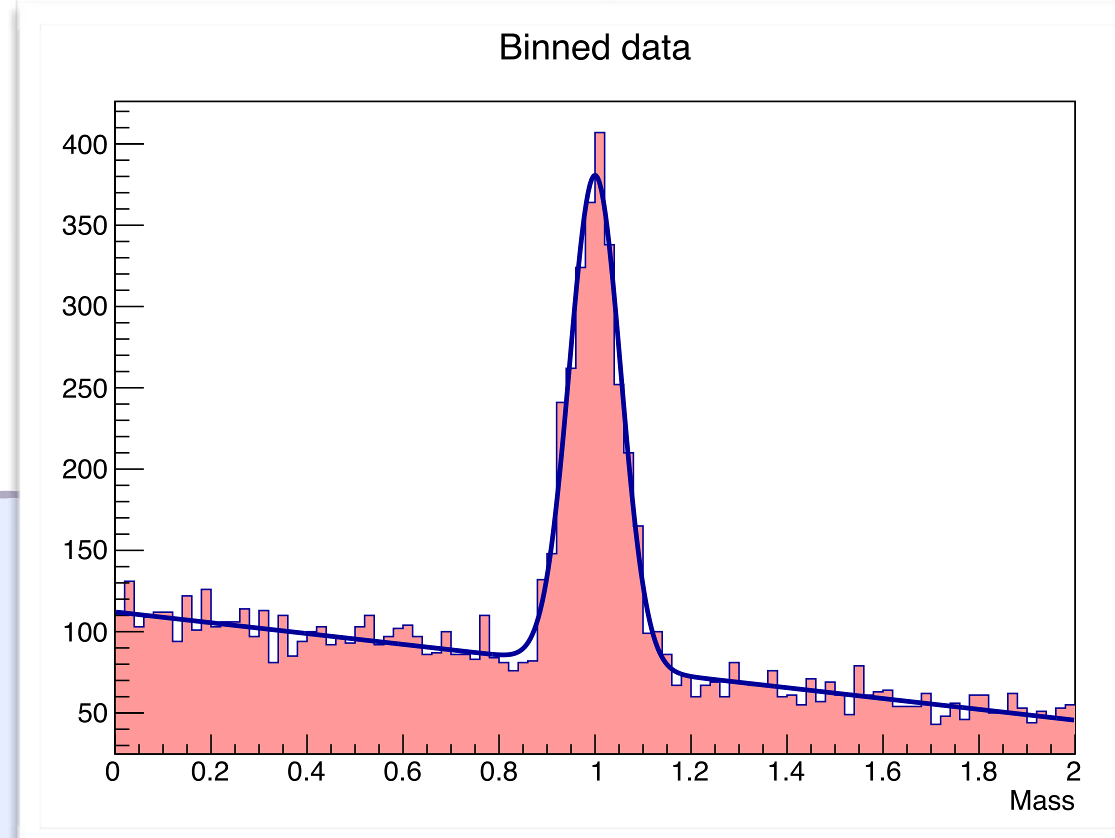
```
void example_05()
{
    TFile *fin = new TFile("example_data.root");
    hist = (TH1D *)fin->Get("hist");

    TMinuit *gMinuit = new TMinuit(5);
    gMinuit->SetFCN(fcn);

    gMinuit->DefineParameter(0, "p0",     100., 1.,    0.,  200.);
    gMinuit->DefineParameter(1, "p1",     -30., 1., -200.,  200.);
    gMinuit->DefineParameter(2, "area",  2000., 1.,    0.,20000.);
    gMinuit->DefineParameter(3, "mean",   1.00, 1.,   0.5,   1.5);
    gMinuit->DefineParameter(4, "width",  0.05, 1., 0.001,  0.15);

    gMinuit->Command("MIGRAD");
    gMinuit->Command("MIGRAD");
    gMinuit->Command("MINOS");

    double par[5],err[5];
    for(int i=0;i<5;i++) gMinuit->GetParameter(i,par[i],err[i]);

    TH1F* curve = new TH1F("curve","curve",hist->GetNbinsX()*5,
         hist->GetXaxis()->GetXmin(),hist->GetXaxis()->GetXmax());

    for(int i=1;i<=curve->GetNbinsX();i++) {
        double x = curve->GetBinCenter(i);
        double f = model(x,par);
        curve->SetBinContent(i,f);
    }
    curve->SetLineWidth(3);

    hist->Draw();
    curve->Draw("csame");
}
```

↖ obtain the output mean & error

Plotting the fitted curve with a histogram with finer bins!

FIT WITH MINUIT (CONT.)

➤ The results should be fully compatible with previous example 04!




```
% root -l example_05.cc
root [0]
Processing example_05.cc...
...
 **********
 **    3 **MINOS
 **********
 FCN=93.6468 FROM MINOS     STATUS=SUCCESSFUL    217 CALLS         364 TOTAL
 EDM=3.78799e-17    STRATEGY= 1  ERROR MATRIX UNCERTAINTY   0.0 per cent
  EXT PARAMETER                  PARABOLIC         MINOS ERRORS
  NO.   NAME      VALUE            ERROR      NEGATIVE      POSITIVE
   1  p0           1.12163e+02   2.00576e+00  -2.00605e+00   2.00576e+00
   2  p1          -3.33062e+01   1.52150e+00  -1.52151e+00   1.52153e+00
   3  area         2.02070e+03   5.83151e+01  -5.82593e+01   5.83721e+01
   4  mean         1.00031e+00   1.70385e-03  -1.70425e-03   1.70425e-03
   5  width        5.34156e-02   1.50983e-03  -1.49782e-03   1.52309e-03
root [1]
```

COMMENTS (III)

- In this example, some coding works are required (*not just 3 lines!*)
- But now we have the **FULL CONTROL** of the fitting process, error definitions, parameters setup, etc.
- Now the function form is defined by ourselves, and one can "read" the area of the signal peak directly. Since it is a "binned fit", one also needs to consider the **bin width**!
- By calling **"MINOS"**, asymmetric errors can be obtained. The difference between the "MIGRAD" and "MINOS" will be discuss in the next lecture.
- However we have to take care of the plot/curve making by ourselves… (*Here a histogram is used to store the curve; this is just a quick and dirty way to produce the plot, and there are other ways to do the same thing.*)

THE LIMITATION OF LEAST-SQUARE METHOD

- Generally you need to produce **histogram(s)** before applying the chi-square fit to your data.
- There are two obvious problems:
  - **The fitting definitely depends on your histogram setup.**
    Many bins → error of each bin could be large/or null bins.
    Fewer bins → loose of resolutions.
  - **Null bins are not defined: no uncertainty can be assigned.**
    (so it cannot work with small number of events…)

Let's examine these two "ill" cases…

TRIAL #1: A MUCH WIDER BIN WIDTH?

➤ Let's re-do the fit with much **wider** bins:

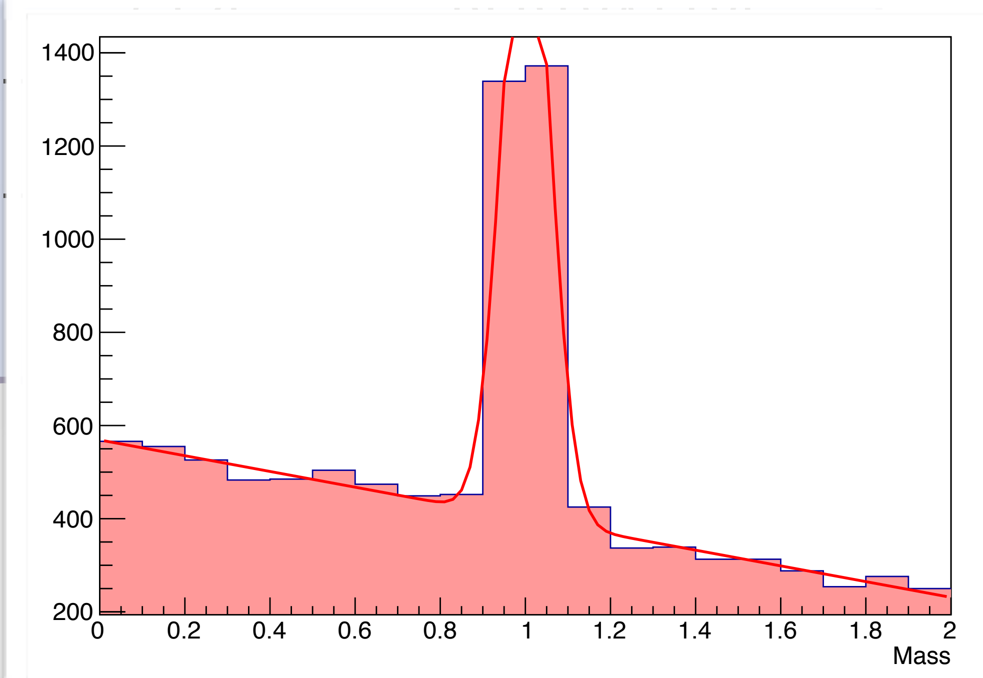
partial example_04b.cc

```
TFile *fin = new TFile("example_data.root");
TNtupleD *nt = (TNtupleD *)fin->Get("nt");
TH1D *hist2 = new TH1D("hist2","Binned data",20,0.,2.);
nt->Project("hist2","mass");
. . .
hist2->Fit("f1");
```

```
FCN=9.47284 FROM MIGRAD    STATUS=CONVERGED      191 CALLS   192 TOTAL
EDM=1.07103e-08    STRATEGY= 1  ERROR MATRIX UNCERTAINTY  0.8 per cent
 EXT PARAMETER                                STEP         FIRST
 NO.   NAME      VALUE            ERROR
  1  p0           5.68959e+02   1.02542e+01
  2  p1          -1.68919e+02   7.66970e+00
  3  p2           3.77367e+01   1.06884e+00
  4  p3           1.00184e+00   1.68374e-03
  5  p4           5.59706e-02   3.44180e-03
```

```
  3  p2           1.73735e+01
  4  p3           1.00031e+00
  5  p4           5.34156e-02
```

(the original fit with 100 bins..)



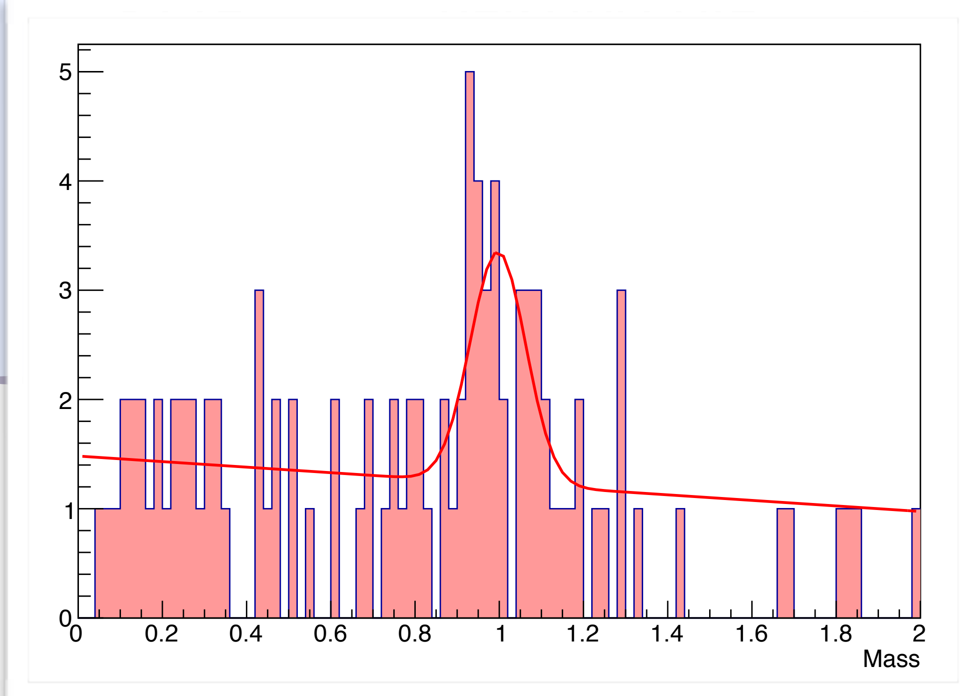
TRIAL #2: WITH MUCH SMALLER SAMPLE?

- Let's re-do the fit with only **1/100** amount of data:

partial **example_04c.cc**

```
TFile *fin = new TFile("example_data.root");
TNtupleD *nt = (TNtupleD *)fin->Get("nt");
TH1D *hist2 = new TH1D("hist2","Binned data",100,0.,2.);
nt->Project("hist2","mass","","",nt->GetEntries()/100);
. . .
hist2->Fit("f1");
```

```
FCN=11.758 FROM MIGRAD    STATUS=CONVERGED     230 CALLS   231 TOTAL
EDM=1.38032e-06     STRATEGY= 1      ERROR MATRIX ACCURATE
 EXT PARAMETER                                  STEP         FIRST
 NO.   NAME      VALUE            ERROR
  1   p0          1.48185e+00   2.90414e-01
  2   p1         -2.53512e-01   2.85730e-01
  3   p2          1.45811e+00   3.01526e-01
  4   p3          9.97050e-01   2.96546e-02
  5   p4          6.57965e-02   2.17127e-02
```



The background level is totally overestimated:

UNBINNED MAXIMUM LIKELIHOOD ESTIMATOR

- In the χ^2 fits, we have to produce histograms first, but for an unbinned maximum likelihood fit, this is not necessary.
- For each event we can have the following likelihood function:

$$L_i = f_s \cdot P_s(x_i; \alpha, \beta) + (1 - f_s) \cdot P_b(x_i; \gamma, \delta)$$

The best solution by maximizing the total likelihood:
Or, by minimizing the value of

$$L = \prod_i^N L_i$$

$$f = -2\ln(L) = -2\sum \log(L_i)$$

Remark: the factor of 2 is very important!

P_s (P_b) : signal (background) PDF
f_s (f_b) : signal (background) fraction
α, β, γ, δ, ... : some fitting parameters to be resolved by the estimator

UNBINNED MAXIMUM LIKELIHOOD ESTIMATOR (CONT.)

- For our simple fitting model in the earlier examples:

$$P_s = G(x;\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left[\frac{-(x-\mu)^2}{2\sigma^2}\right]$$

$$P_b = N[c + b\cdot x] = N'[1 + b'\cdot x]$$

The normalization is very important!

$$\int_0^2 P_b(x)dx = 1 \rightarrow N' = \frac{1}{(2+2\cdot b')}$$

$$L_i = f_s \times \frac{1}{\sqrt{2\pi}\sigma}\exp\left[\frac{-(x_i-\mu)^2}{2\sigma^2}\right] + (1-f_s)\times\left(\frac{1+b'\cdot x}{2+2\cdot b'}\right)$$

➡ With the following floated fitting parameters: f_s, σ, μ, b'

Note: an overall normalization is removed!
only 4 free parameters!

UML FITTER EXAMPLE

partial example_06.cc

```
TH1D *hist = 0;
TNtupleD *nt = 0;
```

↶ take the (unbinned) events from n-tuple instead

```
double model(double x, double *par)
{
    double bprime = par[0];
    double fs     = par[1]/nt->GetEntries();
    double mu     = par[2];
    double sigma  = par[3];

    double norm  = 1./sqrt(2.*TMath::Pi())/sigma;
    double G     = norm*exp(-0.5 * pow((x-mu)/sigma,2));

    return fs * G + (1.-fs) * (1 + bprime*x)/(2. + 2.*bprime);
}
```

The L_i from the previous slide

```
void fcn(int &npar, double *gin, double &f, double *par, int iflag)
{
    f = 0.;
    for (int i=0;i<nt->GetEntries();i++) {
        nt->GetEntry(i);
        double *mass = nt->GetArgs();

        double L = model(*mass,par);

        if (L>0.) f -= 2.*log(L);
        else { f = HUGE; return; }
    }
}
```

↶ $f = -2\ln(L) = -2\sum \log(L_i)$

partial example_06.cc

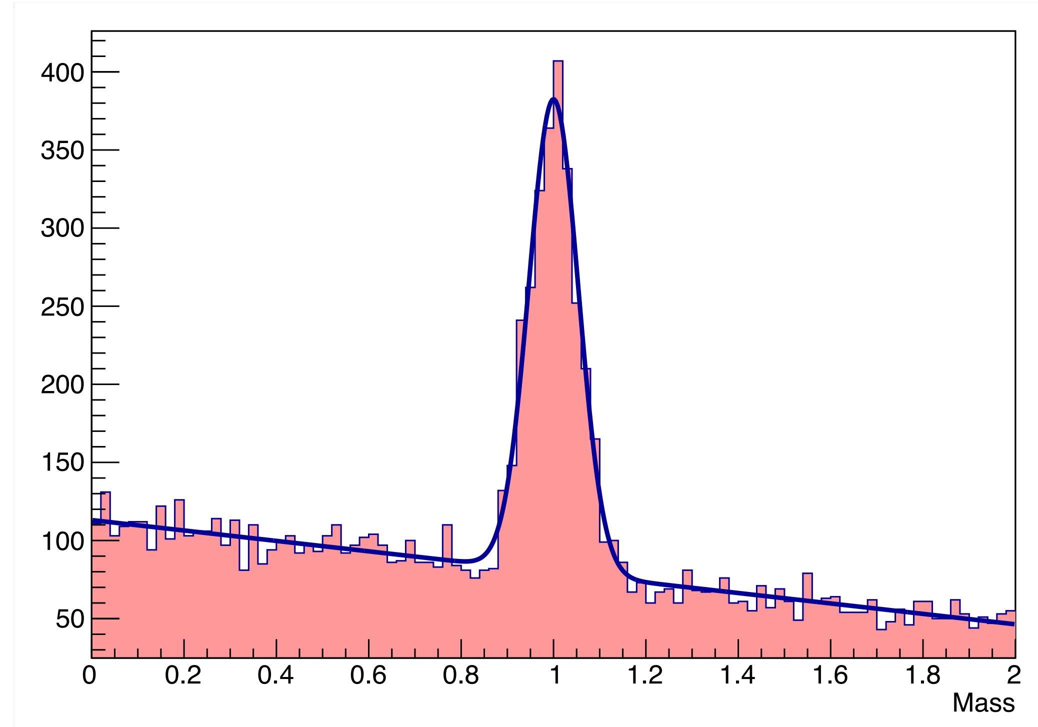
```
void example_06()
{
    TFile *fin = new TFile("example_data.root");
    hist = (TH1D *)fin->Get("hist");
    nt = (TNtupleD *)fin->Get("nt");

    TMinuit *gMinuit = new TMinuit(4);
    gMinuit->SetFCN(fcn);

    gMinuit->DefineParameter(0, "bprime",-0.3, 1.,  -10.,   10.);
    gMinuit->DefineParameter(1, "area", 2000., 1.,    0.,20000.);
    gMinuit->DefineParameter(2, "mean",  1.00, 1.,   0.5,   1.5);
    gMinuit->DefineParameter(3, "width", 0.05, 1., 0.001,  0.15);

    gMinuit->Command("MIGRAD");
    gMinuit->Command("MIGRAD");
    gMinuit->Command("MINOS");

    double par[4],err[4];
    for(int i=0;i<4;i++) gMinuit->GetParameter(i,par[i],err[i]);

    TH1F* curve = new TH1F("curve","curve",hist->GetNbinsX()*5,
         hist->GetXaxis()->GetXmin(),hist->GetXaxis()->GetXmax());

    for(int i=1;i<=curve->GetNbinsX();i++) {
        double x = curve->GetBinCenter(i);
        double f = model(x,par);
        double BinWidth = hist->GetBinWidth(1);
        curve->SetBinContent(i,f*nt->GetEntries()*BinWidth);
    }
    curve->SetLineWidth(3);

    hist->Draw();
    curve->Draw("csame");
}
```

one free parameter is removed; no overall normalization.

↶ the plotting routine

correct for yields & bin width

UML FITTER EXAMPLE (CONT.)



- Just execute the code!

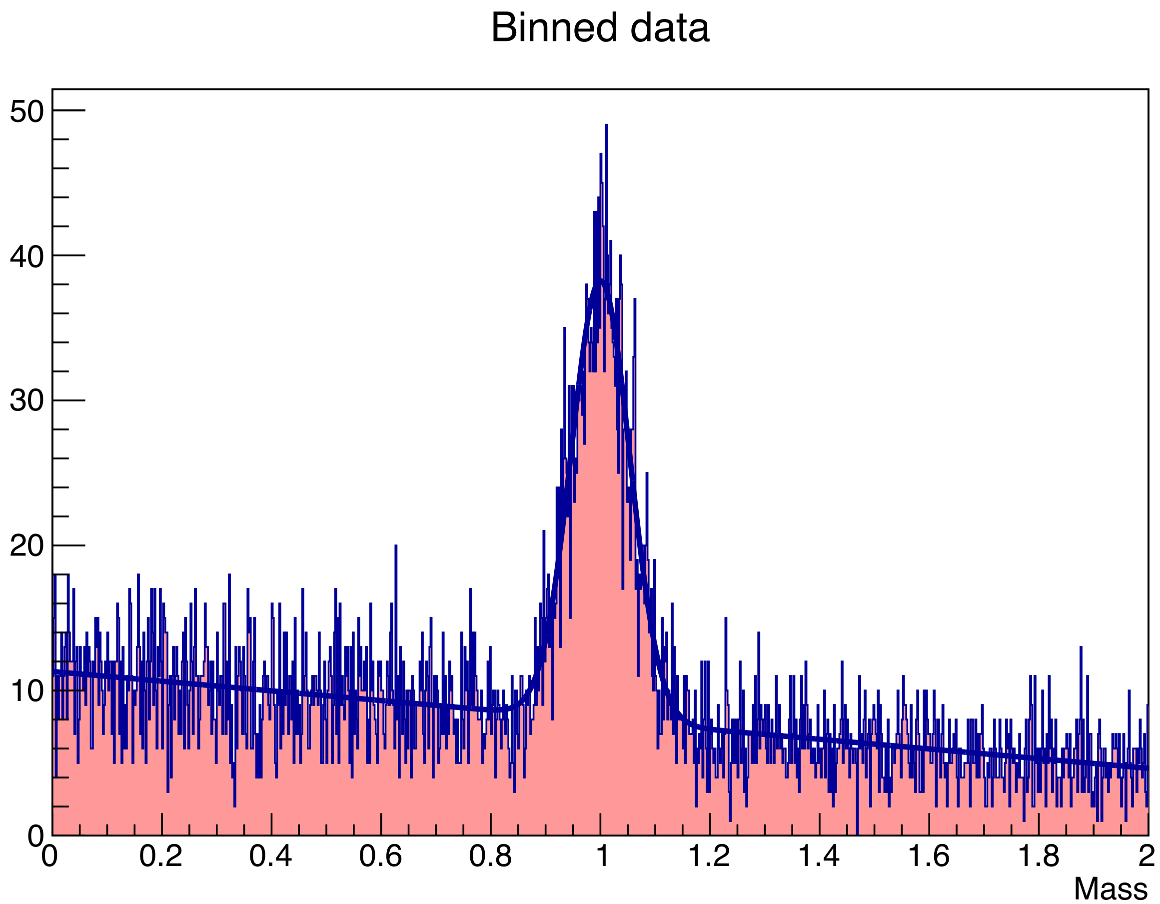
```
FCN=10775.9 FROM MINOS     STATUS=SUCCESSFUL     267 CALLS   376 TOTAL
EDM=2.03171e-09     STRATEGY= 1  ERROR MATRIX UNCERTAINTY   1.8 per cent
 EXT PARAMETER                   PARABOLIC         MINOS ERRORS
 NO.   NAME      VALUE             ERROR      NEGATIVE      POSITIVE
  1  bprime      -2.95047e-01   7.60116e-03  -8.88891e-03   9.19082e-03
  2  area         2.02793e+03   5.45256e+01  -5.44905e+01   5.49236e+01
  3  mean         1.00034e+00   1.80868e-03  -1.69569e-03   1.69695e-03
  4  width        5.34491e-02   1.47157e-03  -1.48382e-03   1.52780e-03
```

- The result should be consistent with the results from the χ^2 fit:

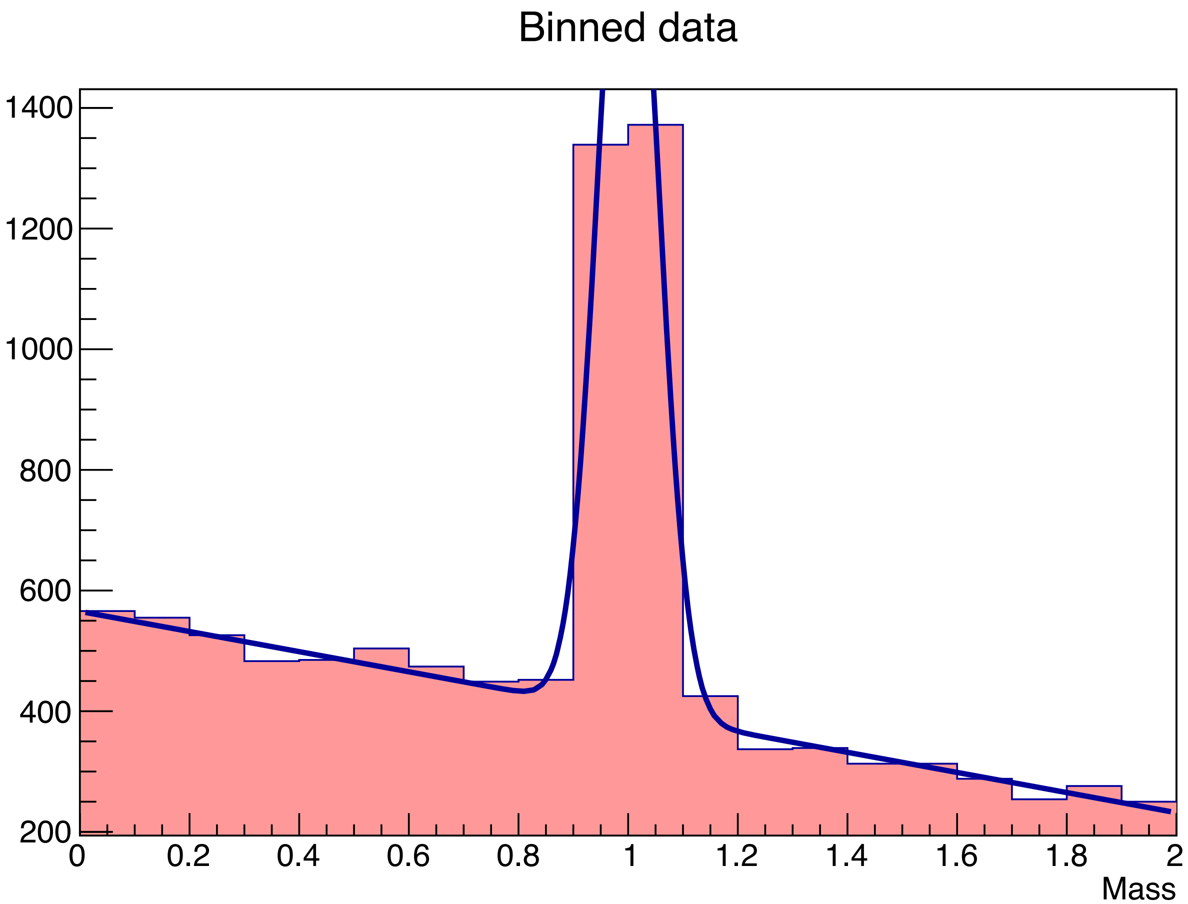
```
  1  p0           1.12163e+02   2.00576e+00  -2.00605e+00   2.00576e+00
  2  p1          -3.33062e+01   1.52150e+00  -1.52151e+00   1.52153e+00
  3  area         2.02070e+03   5.83151e+01  -5.82593e+01   5.83721e+01
  4  mean         1.00031e+00   1.70385e-03  -1.70425e-03   1.70425e-03
  5  width        5.34156e-02   1.50983e-03  -1.49782e-03   1.52309e-03
```

REVISIT THE "PROBLEMS" — BINNING

➤ Now the fit does **NOT** depend on the binning anymore; the binned histogram is ***just for illustration***.



1000 bins



20 bins

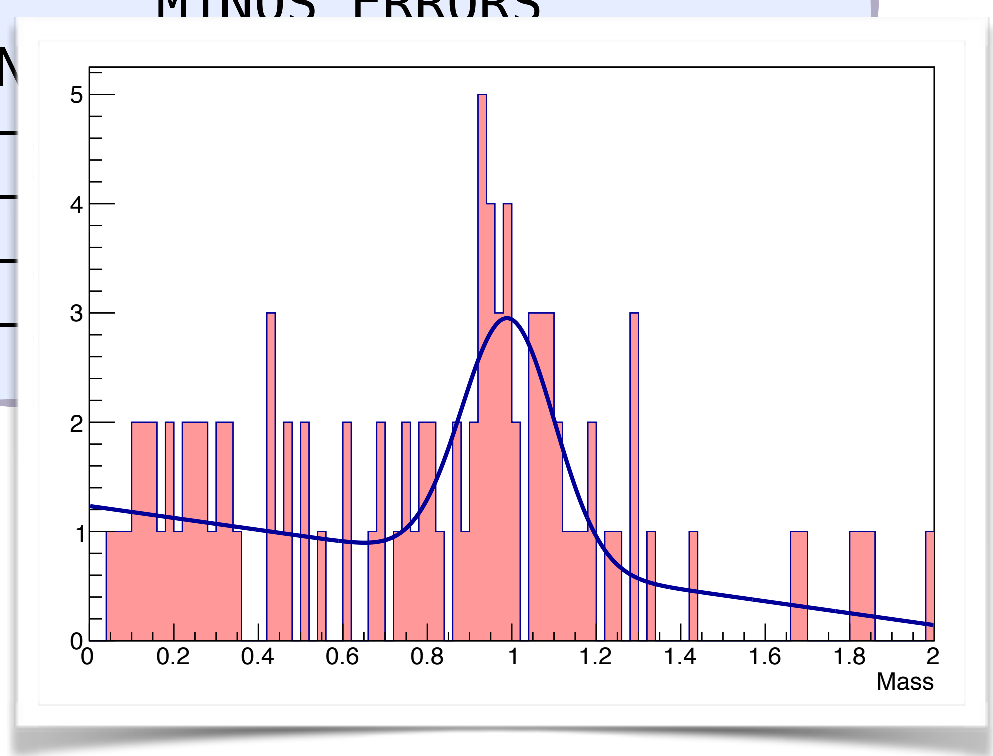
TRIAL #2: WITH MUCH SMALLER SAMPLE?

- Fit with only **1/100** of the data:

partial **example_06b.cc**

```
    for (int i=0;i<nt->GetEntries()/100;i++) {
        nt->GetEntry(i);
        double *mass = nt->GetArgs();

        double L = model(*mass,par);
        if (L>0.) f -= 2.*log(L);
        else { f = HUGE; return; }
```

```
FCN=89.0674 FROM MINOS     STATUS=SUCCESSFUL     286 CALLS    467 TOTAL
EDM=2.36735e-12    STRATEGY= 1   ERROR MATRIX UNCERTAINTY    0.0 per cent
 EXT PARAMETER                   PARABOLIC          MINOS ERRORS
 NO.   NAME       VALUE            ERROR       N
  1   bprime     -4.42016e-01    4.24374e-02   -
  2   area        3.12102e+03    7.81777e+02   -
  3   mean        9.91359e-01    3.00457e-02   -
  4   width       1.10208e-01    3.29341e-02   -
```



The background level is correctly estimated now.

BREAKDOWN OF STANDARD UML FIT

- The unbinned maximum likelihood fit can do the job very well, except for the case of **very few (clean)** events.
- The uncertainty may be underestimated if the background is too small (e.g. one can think of as **$f_s \rightarrow 1$, than error $\rightarrow 0$**).
- Generally there is always a **Poisson error** associated with the total observed events, and it should not be ignored.
- This requires a modification to the likelihood function.

Let's examine following example for such a case first.

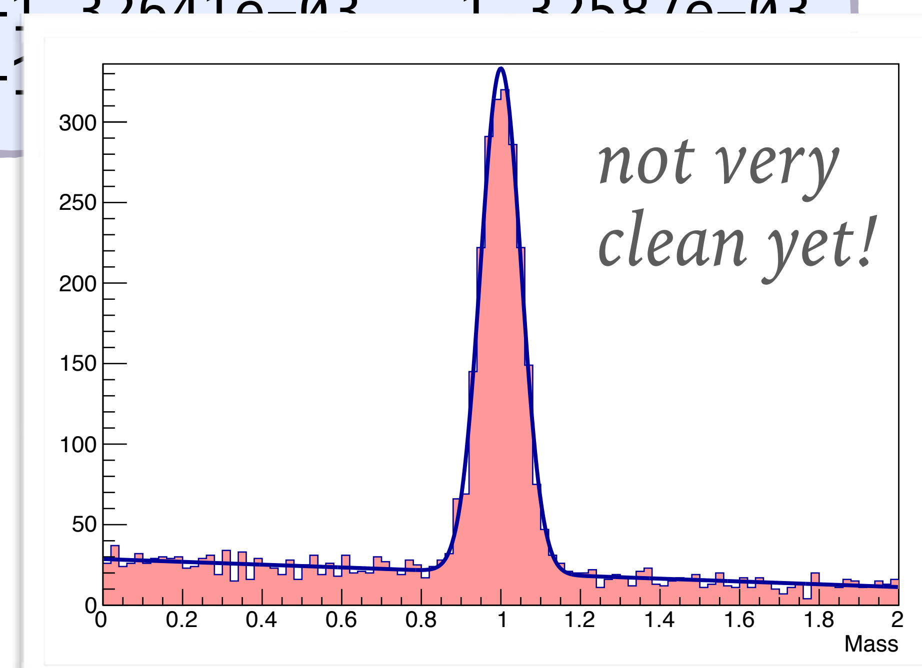
A MUCH CLEANER SAMPLE?

- Just use another example data set with S/N ~ 1:
  http://hep1.phys.ntu.edu.tw/~kfjack/lecture/hepstat/in3/example_data2.root

modify from **example_04.cc**

```
TFile *fin = new TFile("example_data2.root");
```

```
FCN=373.983 FROM MINOS     STATUS=SUCCESSFUL    168 CALLS   255 TOTAL
EDM=8.44189e-15    STRATEGY= 1  ERROR MATRIX UNCERTAINTY   0.0 per cent
 EXT PARAMETER                PARABOLIC         MINOS ERRORS
 NO.   NAME      VALUE          ERROR      NEGATIVE      POSITIVE
  1  bprime     -3.03689e-01   1.74981e-02  -1.69325e-02   1.80906e-02
  2  area        2.00515e+03   3.68918e+01  -3.69279e+01   3.68518e+01
  3  mean        1.00005e+00   1.32596e-03  -1.32641e-03   1.32587e-03
  4  width       5.10625e-02   1.12442e-03  -1
```



- Look at the error of "area", it's actually too small **~1.8%**. The uncertainty should not be smaller then the Poisson error [square-root of the "area", which is **~2.2%**].

THE EXTENDED MAXIMUM LIKELIHOOD ESTIMATOR

- The likelihood function for each event should be modified to

$$L_i = n_s \cdot P_s(x_i; \alpha, \beta) + n_b \cdot P_b(x_i; \gamma, \delta)$$

The best solution by maximizing the total likelihood:

$$L = \frac{\exp[-(n_s + n_b)]}{N!} \prod_i^N L_i$$

Or by minimizing the value of

A constant, can be thrown away.

$$f = -2\ln(L) = 2(n_s + n_b) - 2\sum \log(L_i) - \log(N!)$$

P_s (P_b) : signal (background) PDF

n_s (n_b) : signal (background) yields

α, β, γ, δ, ... : some fitting parameters to be resolved by the estimator

EXTENDED UML FITTER EXAMPLE

partial example_07.cc

```
double model(double x, double *par)
 {
    double bprime = par[0];
    double fs     = par[1]/nt->GetEntries();
    double mu     = par[2];
    double sigma  = par[3];

    double norm  = 1./sqrt(2.*TMath::Pi())/sigma;
    double G     = norm*exp(-0.5 * pow((x-mu)/sigma,2));

    return ns * G + nb * (1 + bprime*x)/(2. + 2.*bprime);
 }

 void fcn(int &npar, double *gin, double &f, double *par, int iflag)
 {
    double nb     = par[0];
    double ns     = par[2];
    f = 2.*(ns+nb);
    for (int i=0;i<nt->GetEntries();i++) {
        nt->GetEntry(i);
        double *mass = nt->GetArgs();

        double L = model(*mass,par);

        if (L>0.) f -= 2.*log(L);
        else { f = HUGE; return; }
    }
 }
```

The updated L_i, note the yields in front of the PDF

$$f = 2(n_s + n_b) - 2\sum \log(L_i)$$

partial example_07.cc

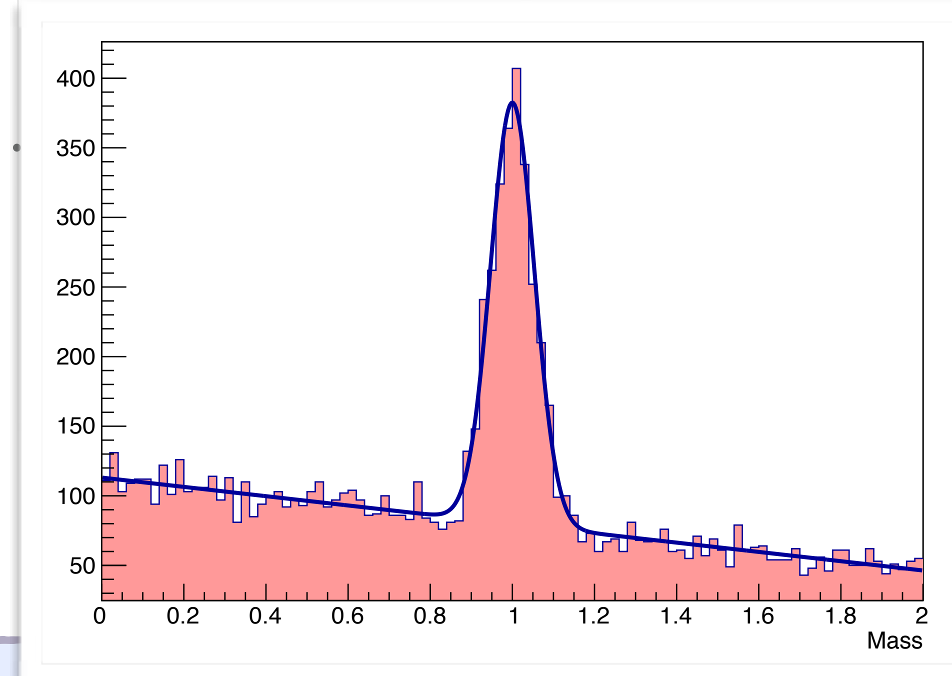
```
void example_07()
{
    TFile *fin = new TFile("example_data.root");
    hist = (TH1D *)fin->Get("hist");
    nt = (TNtupleD *)fin->Get("nt");

    TMinuit *gMinuit = new TMinuit(5);
    gMinuit->SetFCN(fcn);

    gMinuit->DefineParameter(0, "nbkg", 8000., 1.,    0.,20000.);
    gMinuit->DefineParameter(1, "bprime",-0.3, 1.,  -10.,   10.);
    gMinuit->DefineParameter(2, "area", 2000., 1.,    0.,20000.);
    gMinuit->DefineParameter(3, "mean",  1.00, 1.,   0.5,   1.5);
    gMinuit->DefineParameter(4, "width", 0.05, 1., 0.001,  0.15);

    gMinuit->Command("MIGRAD");
    gMinuit->Command("MIGRAD");
    gMinuit->Command("MINOS");

    double par[5],err[5];
    for(int i=0;i<5;i++) gMinuit->GetParameter(i,par[i],err[i]);

    TH1F* curve = new TH1F("curve","curve",hist->GetNbinsX()*5,
        hist->GetXaxis()->GetXmin(),hist->GetXaxis()->GetXmax());

    for(int i=1;i<=curve->GetNbinsX();i++) {
        double x = curve->GetBinCenter(i);
        double f = model(x,par);
        double BinWidth = hist->GetBinWidth(1);
        curve->SetBinContent(i,f*BinWidth);
    }
    curve->SetLineWidth(3);

    hist->Draw();
    curve->Draw("csame");
}
```

Now the total yield is not fixed anymore!

correct for bin width only

EXTENDED UML FITTER EXAMPLE (II)

Remark: you may find the error increases a little bit!



- Let's just try it:

```
FCN=-153431 FROM MINOS     STATUS=SUCCESSFUL    426 CALLS   560 TOTAL
EDM=4.96548e-07    STRATEGY= 1  ERROR MATRIX UNCERTAINTY   2.2 per cent
 EXT PARAMETER                  PARABOLIC          MINOS ERRORS
 NO.   NAME      VALUE            ERROR      NEGATIVE      POSITIVE
  1  nbkg        7.97207e+03   9.69213e+01  -9.63220e+01   9.70525e+01
  2  bprime     -2.95047e-01   7.41395e-03  -8.88851e-03   9.19127e-03
  3  area        2.02793e+03   5.77095e+01  -5.79722e+01   5.87229e+01
  4  mean        1.00034e+00   1.72936e-03  -1.69575e-03   1.69689e-03
  5  width       5.34492e-02   1.44846e-03  -1.48393e-03   1.52770e-03
```

- The result is (*almost*) the same as the standard UML fit:

```
  1  bprime     -2.95047e-01   7.60116e-03  -8.88891e-03   9.19082e-03
  2  area        2.02793e+03   5.45256e+01  -5.44905e+01   5.49236e+01
  3  mean        1.00034e+00   1.80868e-03  -1.69569e-03   1.69695e-03
  4  width       5.34491e-02   1.47157e-03  -1.48382e-03   1.52780e-03
```

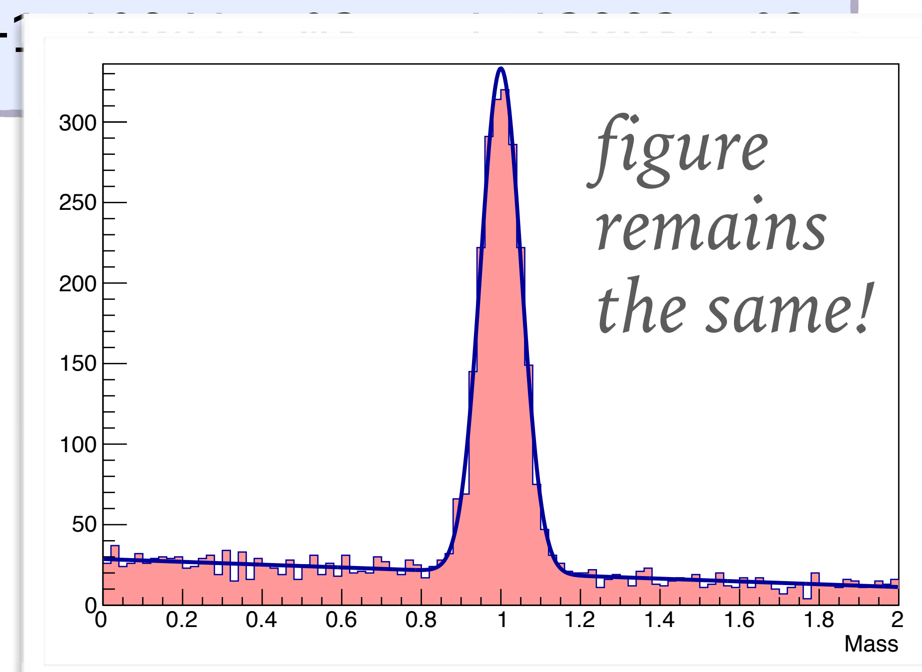
EXTENDED UML FITTER EXAMPLE (III)

➤ Try the “cleaner” example data:

modify from **example_07.cc**

```
TFile *fin = new TFile("example_data2.root");
```

```
FCN=-57978.4 FROM MINOS     STATUS=SUCCESSFUL     389 CALLS 537 TOTAL
EDM=1.66126e-07     STRATEGY= 1   ERROR MATRIX UNCERTAINTY   2.1 per cent
 EXT PARAMETER                    PARABOLIC         MINOS ERRORS
 NO.   NAME      VALUE              ERROR        NEGATIVE      POSITIVE
  1   nbkg        1.99485e+03   4.91563e+01  -4.81594e+01   4.89190e+01
  2   bprime     -3.03687e-01   1.66553e-02  -1.69340e-02   1.80908e-02
  3   area        2.00515e+03   4.85967e+01  -4.83070e+01   4.89816e+01
  4   mean        1.00005e+00   1.33072e-03  -1.32641e-03   1.32587e-03
  5   width       5.10626e-02   1.16962e-03  -1
```



➤ Since this extended ML fit includes the Poisson error to the total # of events, the uncertainties can be correctly estimated.

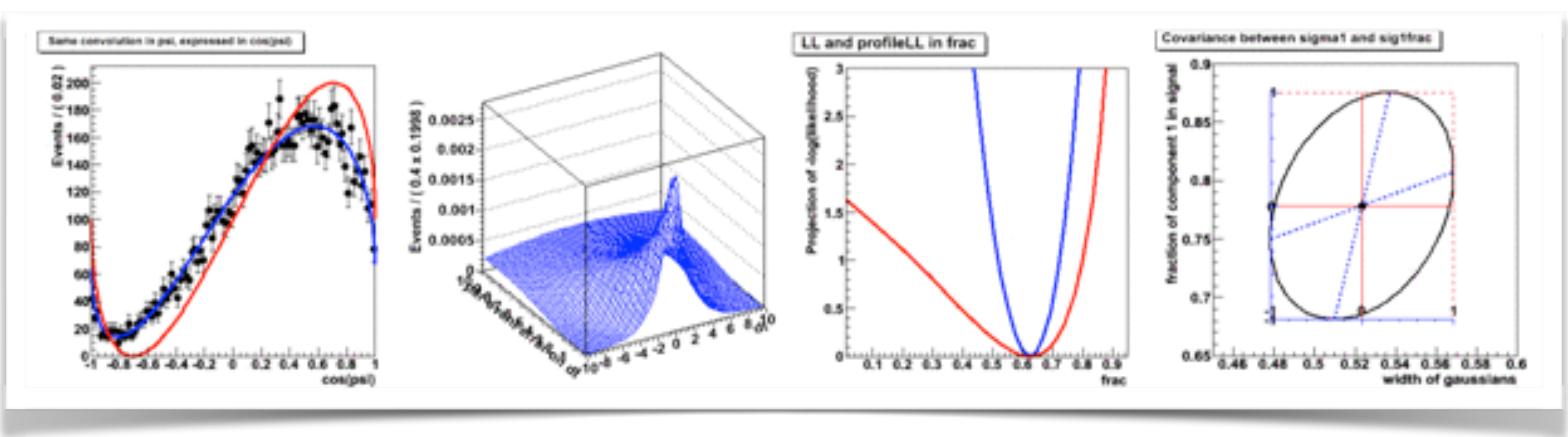
➤ Now the error of “area” is right **(>2.2%)**.

Going through our first step toward RooFit

ROOFIT: INTRODUCTION

Remark:
RooFit ≠ Root Fit

- The **RooFit** library provides a toolkit for modeling the expected distribution of events in a physics analysis.
- Models can be used to perform **unbinned maximum likelihood fits**, **produce plots**, and **generate toy Monte Carlo** samples for various studies.
- RooFit was originally developed for the BaBar collaboration. The software is primarily designed as a particle physics data analysis tool, but its general nature and open architecture make it useful for other types of data analysis also.



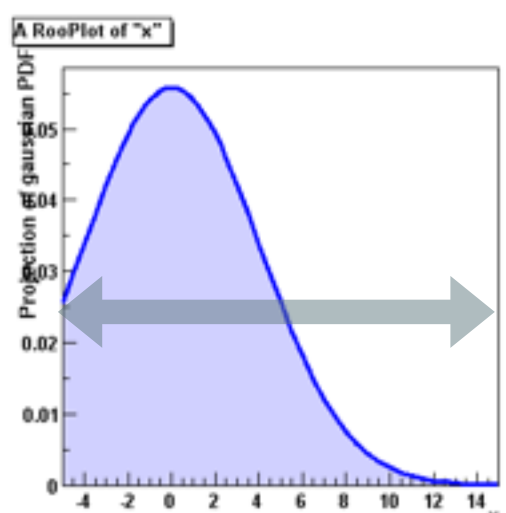
ROOFIT: PROBABILITY DENSITY FUNCTIONS

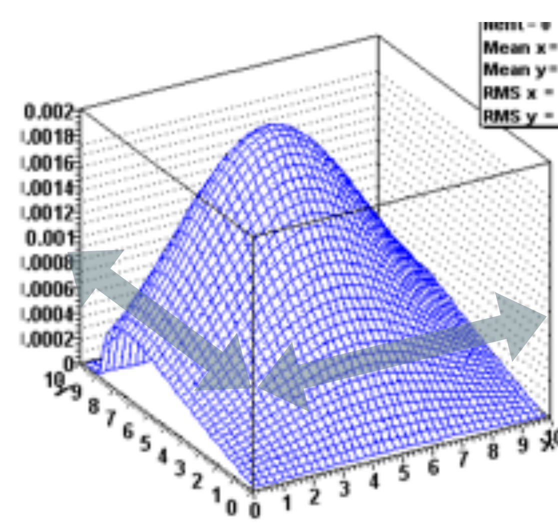
- The important/fundamental property of any probability density function f(X|θ):

$$\int_{\Omega} f(X|\theta)dx = 1$$

- It is relatively easy to construct PDF in 1D; generally requires (*much*) more effort for hight dimensions.
- RooFit automatically takes care of this (within some limitations, of course)
- User supplied function does need not be normalized beforehand.

$$\int_{x_1}^{x_2} f(x|\theta)dx = 1$$





$$\int_{x_1}^{x_2}\int_{y_1}^{y_2} f(x,y|\theta)dxdy = 1$$

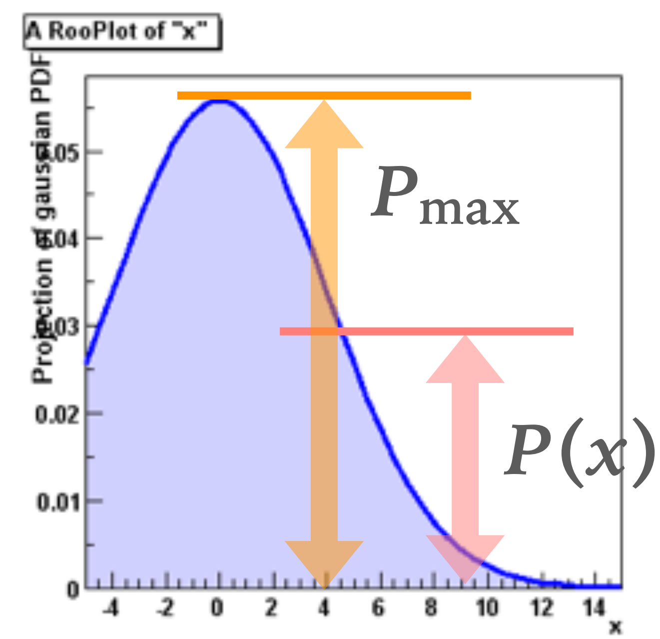
ROOFIT: LIKELIHOOD FIT & RANDOM DISTRIBUTION GENERATION

- **Perform likelihood fit in a single call**
  - Construct likelihood function based on the PDF objects.
  - Bind data and PDF objects together
  - Solve the parameters by minimizing the **–ln(*L*)** using Minut/Minut2/etc. engines.

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

$$-\ln(L) = -\sum_{i=1}^{N} \ln f(X_i|\theta)$$

- **Perform random distribution generation**
  - Mostly automatic: as the PDF built, the random variables can be generated!
  - Automatically choice of methods: Accept/reject method, 'Direct' generation, etc.



***Accept:* $P(x) < P_{max}$**

ROOFIT: CORE DESIGN

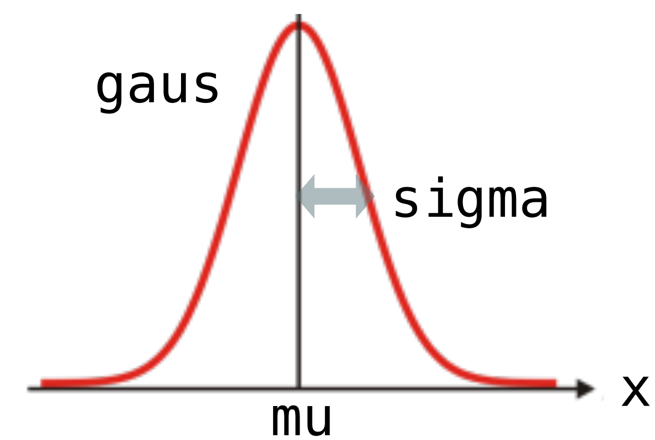
➤ RooFit introduces a granular structure in its mapping of mathematical data models components to C++ objects.

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`

There are plenty of RooXXX classes!

CONSTRUCT ROOFIT OBJECTS

- In RooFit every variable, datapoint, function, PDF are all represented as C++ objects.
- Objects are classified by the data/function type they are representing instead of by their roles in a particular setup.
- Example construction of a Gaussian PDF:



object name (in your code) · object name (in ROOT record) · title · range (min,max) · initial, min, max

```
{
    RooRealVar x("x", "random variable", 0.0, 1.0);

    RooRealVar mu("mu", "mean parameter", 0.5, 0.0, 1.0);
    RooRealVar sigma("sigma", "width parameter", 0.1, 0.0, 0.3);

    RooGaussian gaus("gaus", "Gaussian PDF", x, mu, sigma);
}
```

example_08.cc

references of the objects

BASIC OPERATIONS OF ROOFIT OBJECTS

➤ Print value and attributes

```
root [1] x.Print()
RooRealVar::x = 0.5  L(0 - 1)
```

➤ Assign new value

```
root [1] x.setVal(0.4)
```

➤ Retrieve contents

```
root [4] double x_val = x.getVal()
(double) 0.400000
```

➤ Retrieve contents from depending objects

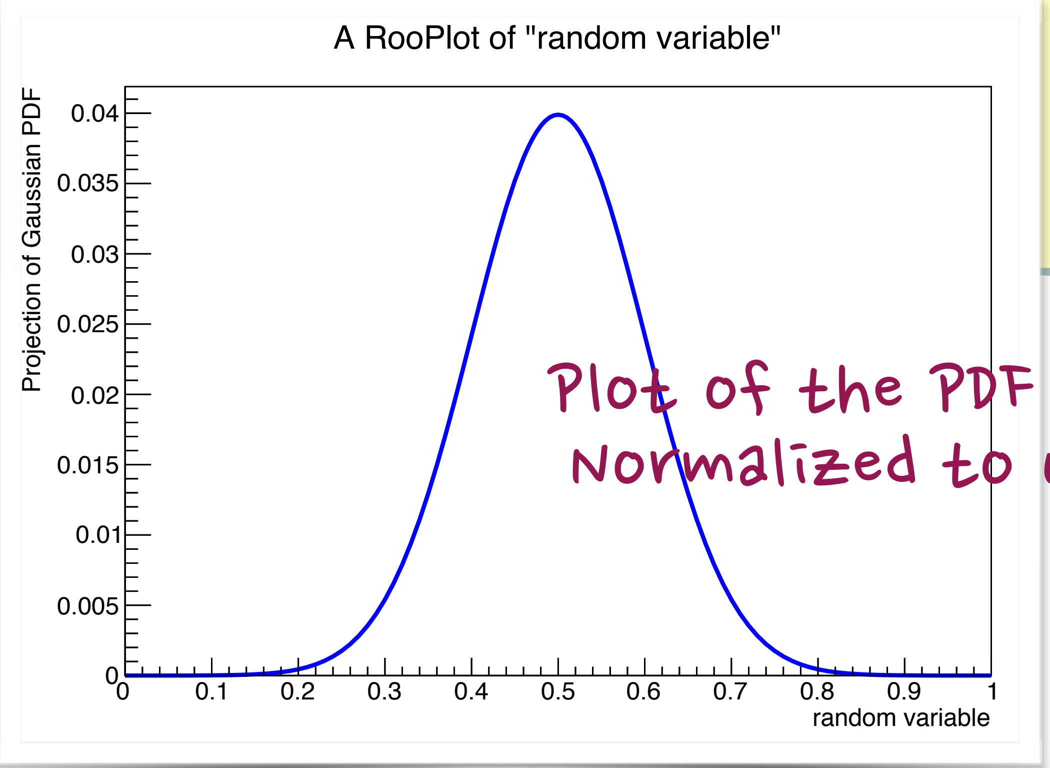
```
root [5] gaus.getVal()
(double) 0.606531
root [6] x.setVal(0.8)
root [7] gaus.getVal()  ↶ automatically updating related objects!
(double) 0.011109
```

BASIC OPERATIONS OF A PDF

➤ Construct a Gaussian PDF and make a plot!

example_08a.cc

```
{
    // observable
    RooRealVar x("x", "random variable", 0.0, 1.0);

    // parameters
    RooRealVar mu("mu", "mean parameter", 0.5, 0.0, 1.0);
    RooRealVar sigma("sigma", "width parameter", 0.1, 0.0, 0.3);

    // PDF
    RooGaussian gaus("gaus", "Gaussian PDF", x, mu, sigma);

    // plot the PDF
    RooPlot* frame = x.frame();
    gaus.plotOn(frame);
    frame->Draw();
}
```



Plot of the PDF itself
Normalized to unity

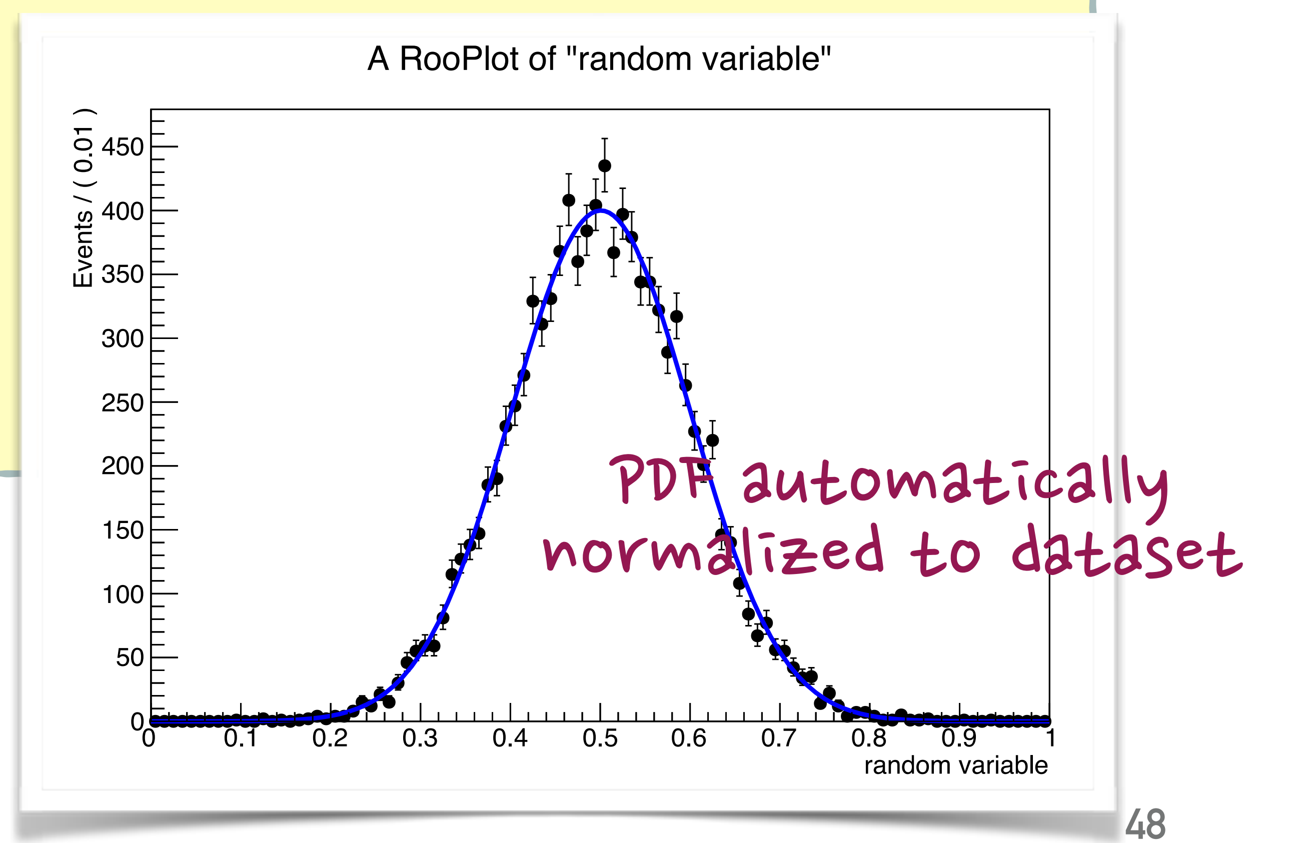
A RooPlot is an empty frame capable of holding anything plotted versus it variable

BASIC OPERATIONS OF A PDF (CONT.)

➤ PDF construction + event generation + fit + plot

example_08b.cc

```
{
    // observable/parameter/PDF construction
    RooRealVar x("x", "random variable", 0.0, 1.0);
    RooRealVar mu("mu", "mean parameter", 0.5, 0.0, 1.0);
    RooRealVar sigma("sigma", "width parameter", 0.1, 0.0, 0.3);
    RooGaussian gaus("gaus", "Gaussian PDF", x, mu, sigma);

    // Generate a random set
    RooDataSet* data = gaus.generate(x,10000);

    // Fit pdf to the random data
    gaus.fitTo(*data);

    // plot the data & PDF
    RooPlot* frame = x.frame();
    data->plotOn(frame);
    gaus.plotOn(frame);
    frame->Draw();
}
```



Once the model is built, all of the rest operations are relatively simple!

OBSERVABLE VERSUS PARAMETERS

- The RooFit PDF objects have no intrinsic notion difference for a variable treated as a parameter or an observable:

```
RooGaussian gaus("gaus", "Gaussian PDF", x, mu, sigma);
```

- However the normalization depends on whether parameter/observable interpretation of variables:

$$\int_\Omega f(X|\theta)dx = 1$$

X: observables
θ: parameters

- Parameter/observable interpretation is automatic/implicit when a PDF is used together with a dataset
  - All variables as the member of a dataset are **observables**
  - The rest of variables are **parameters**
  - Lower/upper bounds are the normalization range if variable is observable, or the Minuit bounds if the variable is a parameter.

LISTS VERSUS SETS

- RooFit has two (*confusing?*) collection classes that are frequently passed as arguments or returned as argument.
- Set semantics — **RooArgSet**
  - Each element can appear only once.
  - No actual ordering of the elements.

```
RooArgSet s1(x,y,z);
RooArgSet s2(x,x,y); // Wrong!!
```

- List semantics — **RooArgList**
  - Elements may be inserted multiple times into the same list.
  - Insertion order is preserved.

```
RooArgList l1(x,y,z);
RooArgList l2(x,x,y);
l2.Print();
```

```
RooArgList:::
  1) RooRealVar::x: "x"
  2) RooRealVar::x: "x"
  3) RooRealVar::y: "y"
```

BUILDING YOUR PDF MODELS

- RooFit already provides a good collection of many commonly used PDFs. Can be used as “building blocks”:

RooArgusBG	Argus background shape
RooBCPEffDecay	B^0 decay with CP violation
RooBMixDecay	B^0 decay with mixing
RooBifurGauss	Bifurcated Gaussian
RooBreitWigner	Breit-Wigner shape
RooCBShape	Crystal Ball function
RooChebychev	Chebychev polynomial
RooDecay	Simple decay function
RooDircPdf	DIRC resolution description
RooDstD0BG	D^* background description
RooExponential	Exponential function
RooGaussian	Gaussian function
RooKeysPdf	Non-parametric data description
Roo2DKeysPdf	Non-parametric data description (2D)
RooPolynomial	Generic polynomial PDF
RooVoigtian	Breit-Wigner convoluted w/ Gaussian

And many, many others…

GENERIC EXPRESSION PDFS

- If your favorite PDF isn't there and you don't want to code a PDF class right away (*although sometimes it runs faster*):
  - you may try the **RooGenericPdf**
- Just write down the PDFs expression as we did for root formula class:

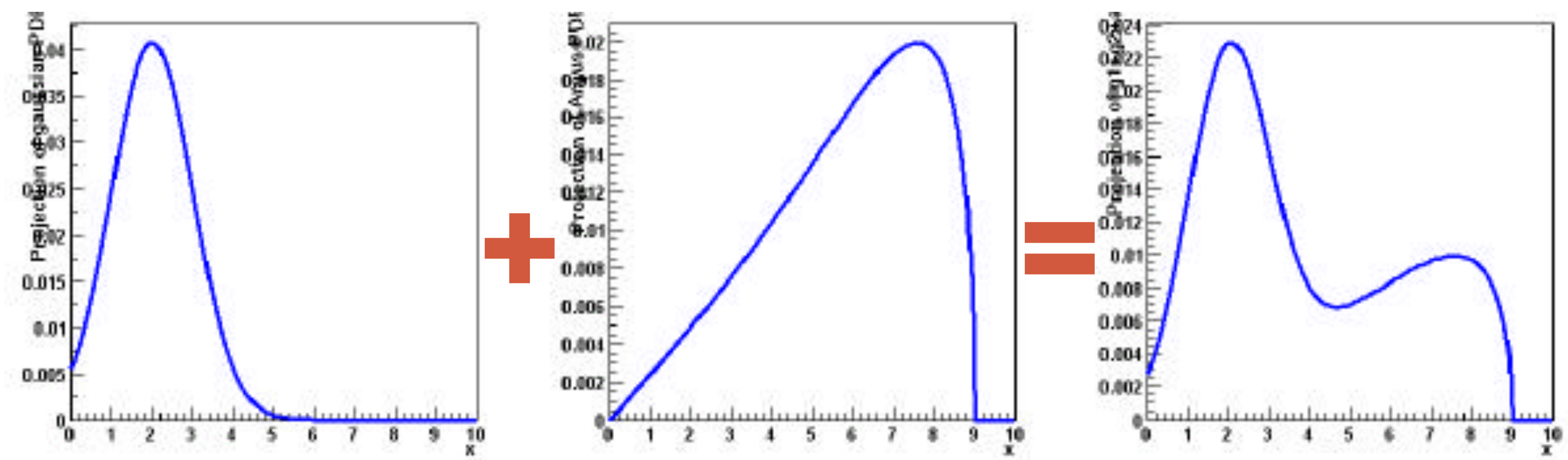
```
// PDF variables
RooRealVar x("x","x",-10.,10.);
RooRealVar y("y","y",0,5);
RooRealVar a("a","a",3.0);
RooRealVar b("b","b",-2.0);

// Generic PDF
RooGenericPdf model("model","Generic PDF",
                    "exp(x*y+a)-b*x",RooArgSet(x,y,a,b));
```

- The normalization will be taken cared automatically with numerical integration.

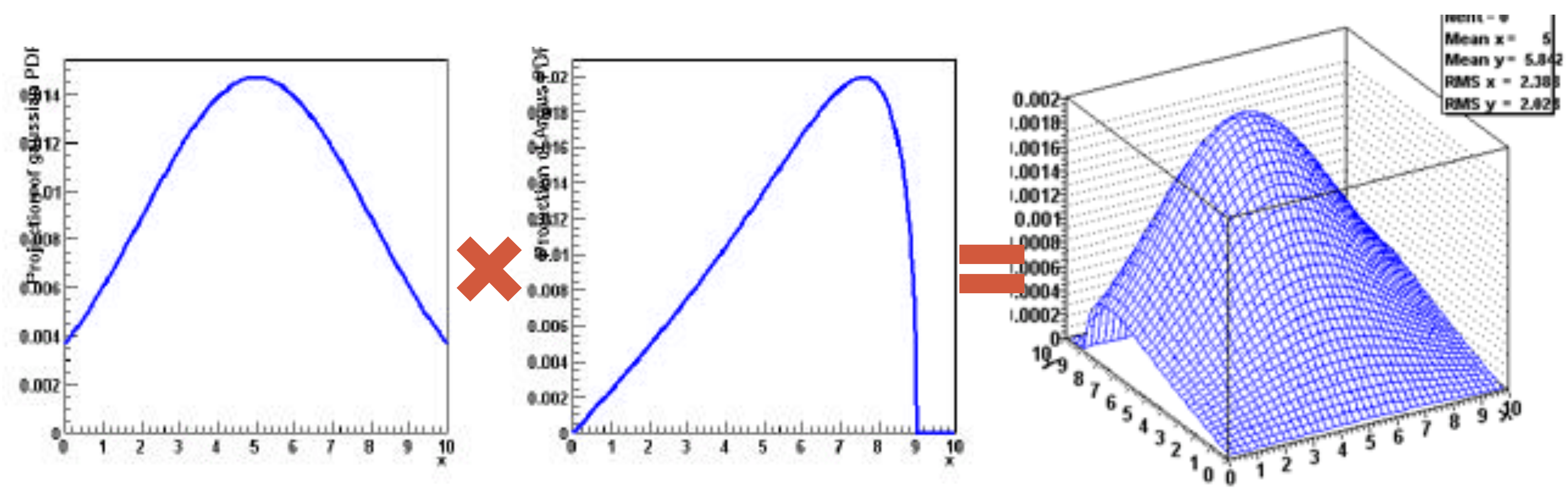
BUILD MODELS WITH BLOCKS

- Complex PDFs be can be easily composed using the operator classes:

RooAddPdf as addition



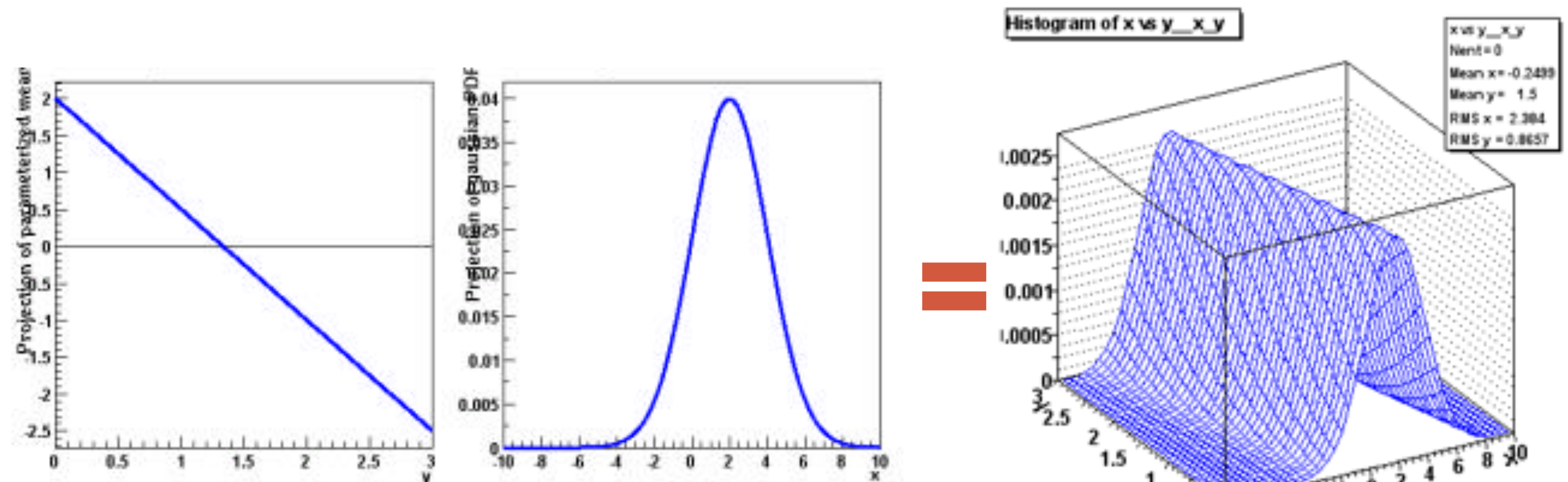
RooProdPdf as multiplication



BUILD MODELS WITH BLOCKS

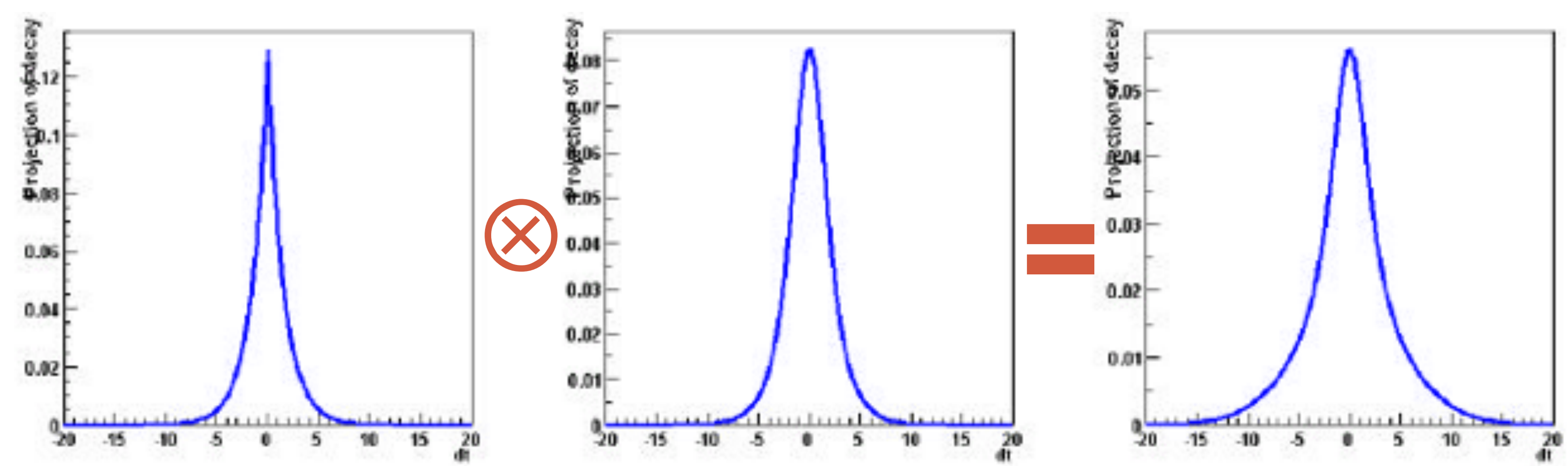
➤ Complex PDFs be can be easily composed using the operator classes:

Composition (parameter dependence)



$m(y; a_0, a_1)$ $g(x; m, s)$ = $g(x, y; a_0, a_1, s)$

Convolution

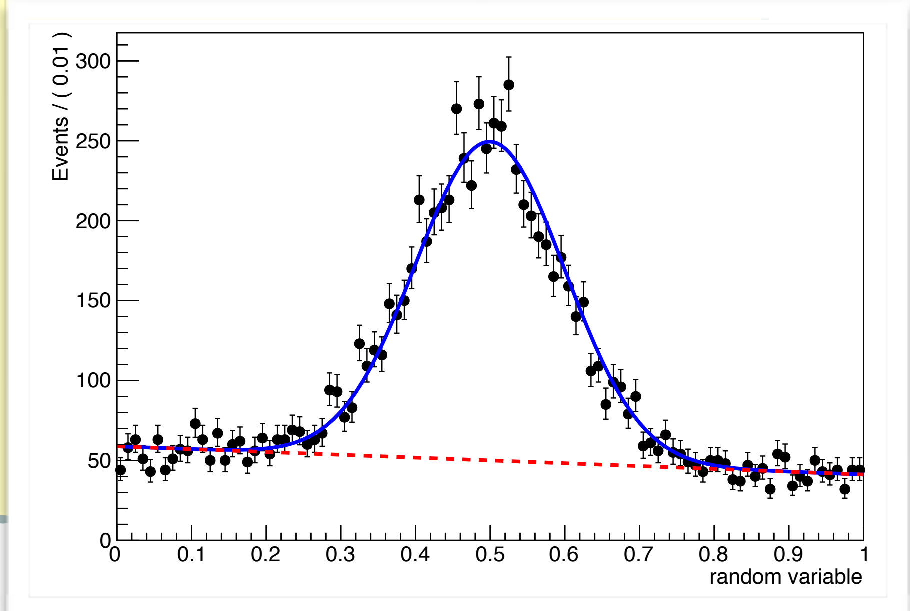


ADD PDF EXAMPLE

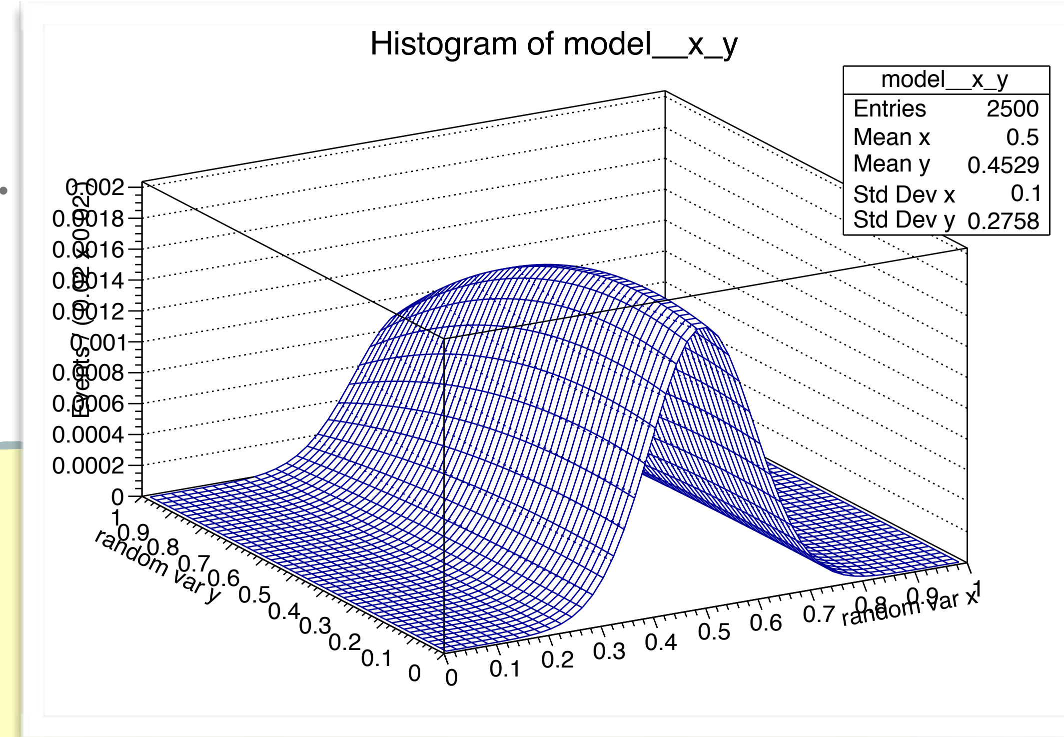
$$P(x;\mu,\sigma,s) = f\cdot G(x;\mu,\sigma) + (1-f)\cdot N\cdot(1+s\cdot x)$$

example_09.cc

```
using namespace RooFit;

// observable
RooRealVar x("x", "random variable", 0.0, 1.0);

// Gaussian model
RooRealVar mu("mu", "mean parameter", 0.5, 0.0, 1.0);
RooRealVar sigma("sigma", "width parameter", 0.1, 0.0, 0.3);
RooGaussian gaus("gaus", "Gaussian PDF", x, mu, sigma);

// Linear function: 1 + slope*x
RooRealVar slope("slope", "slope param.", -0.3, -10., 10.);
RooPolynomial linear("linear","Linear func.",x,RooArgSet(slope));

// add up: Gaussian + linear
RooRealVar fraction("fraction","fraction of Gaussian",0.5,0.,1.);
RooAddPdf model("model","PDF model",
    RooArgList(gaus,linear), RooArgList(fraction));

// generate random data, plot
RooDataSet *data = model.generate(x, 10000);
RooPlot* frame = x.frame();
data->plotOn(frame);
model.plotOn(frame);
model.plotOn(frame,Components(linear),
    LineStyle(7),LineColor(kRed));
frame->Draw();
```



PRODUCT PDF EXAMPLE



example_10.cc

```
{
    // observables
    RooRealVar x("x", "var x", 0.0, 1.0);
    RooRealVar y("y", "var y", 0.0, 1.0);

    // Gaussian model of x
    RooRealVar mu("mu", "mean parameter", 0.5, 0.0, 1.0);
    RooRealVar sigma("sigma", "width parameter", 0.1, 0.0, 0.3);
    RooGaussian gaus("gaus", "Gaussian PDF", x, mu, sigma);

    // Polynomial of y: 1 + p1*y + p2*y^2
    RooRealVar p1("p1", "coeff. of y^1", +0.3, -10., 10.);
    RooRealVar p2("p2", "coeff. of y^2", -0.8, -10., 10.);
    RooPolynomial poly("poly",
        "2nd order poly.", y, RooArgSet(p1,p2));

    // product: Gaussian(x) * Polynomial(y)
    RooProdPdf model("model","PDF model", gaus,poly);

    TH1* model_hist = model.createHistogram("x,y",50,50);
    model_hist->Draw("surf");
}
```

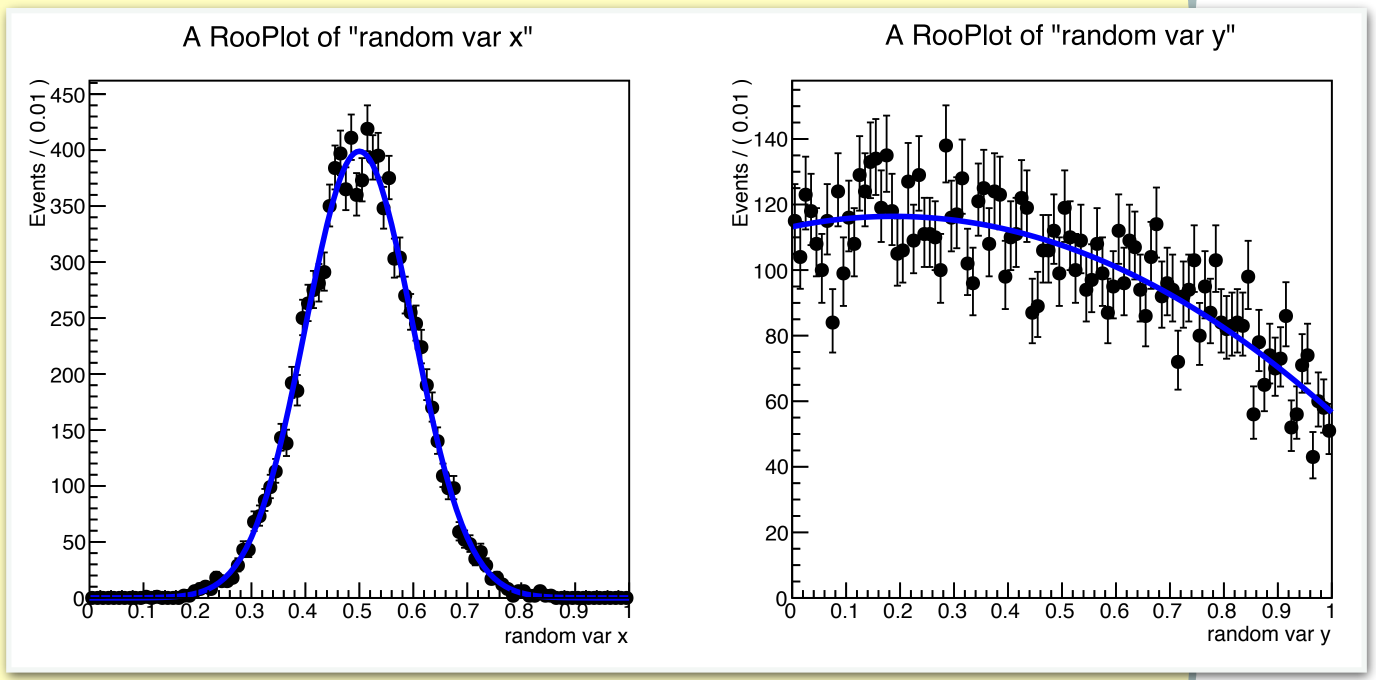
$$P(x,y;\mu,\sigma,p_1,p_2) = G(x;\mu,\sigma) \times N \cdot (1 + p_1 \cdot y + p_2 \cdot y^2)$$

2D DATA AND PROJECTION

- In fact it requires some effort to produce correct 2D to 1D projections of the PDF (through **conditional PDFs**):
  - Plotting a dataset **(*x*,y) versus *x*** represents a **projection over *y***.
  - To overlay PDF(x,y), one must integrate $\int \mathrm{PDF}(x,y)dy$
- RooFit takes care of this automatically.

example_10a.cc

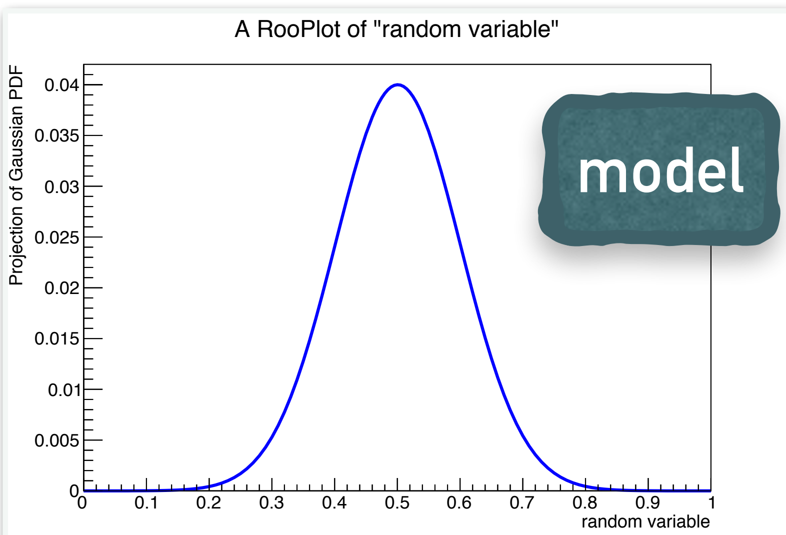
```
// generate random data, plot
RooDataSet *data = model.generate(RooArgSet(x,y), 10000);
RooPlot* frame_x = x.frame();
RooPlot* frame_y = y.frame();

TCanvas *c1 =
  new TCanvas("c1","Canvas");
c1->Divide(2);

c1->cd(1);
data->plotOn(frame_x);
model.plotOn(frame_x);
frame_x->Draw();

c1->cd(2);
data->plotOn(frame_y);
model.plotOn(frame_y);
frame_y->Draw();
```



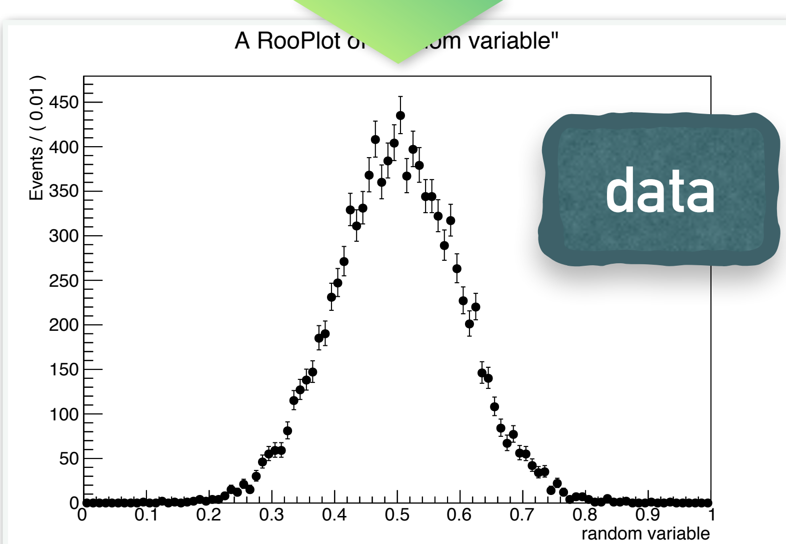
$f(x) = \int \mathrm{PDF}(x,y)dy \qquad f(y) = \int \mathrm{PDF}(x,y)dx$

GENERATION & FITTING

➤ Such an operation actually works for any PDF model:



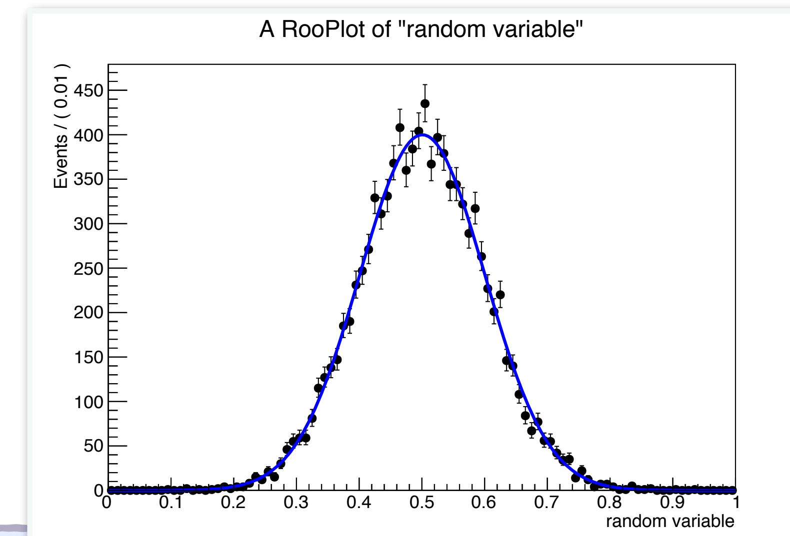

```
model.generate(x,10000);
```




```
RooFitResult *res = model.fitTo(*data, ...);
```

```
COVARIANCE MATRIX CALCULATED SUCCESSFULLY
FCN=-8863.01 FROM HESSE STATUS=OK 10 CALLS 34 TOTAL
EDM=1.57332e-06  STRATEGY= 1  ERROR MATRIX ACCURATE
EXT PARAMETER
NO.   NAME       VALUE              ERROR
 1   mu          5.00665e-01   9.97372e-04
 2   sigma       9.97366e-02   7.05314e-04
```

```
RooPlot* fm = x.frame();
data->plotOn(fm);
model.plotOn(fm);
fm->Draw();
```



GENERATION & FITTING (CONT.)

➤ Some useful options can be added to "generate" and "fitTo" to the "..." part:

```
RooDataSet* data = model.generate(x, ...);
```

Extended(flag)	The actual number of events generated will be sampled from a Poisson distribution with $\mu=N_{evt}$.
ProtoData(data)	Specified existing dataset as a prototype: some of the variables will be used in the generation.

```
RooFitResult *res = model.fitTo(*data, ...);
```

Minimizer(type,algo)	Choose minimization package and algorithm to use.
SumW2Error(flag)	Apply correction to errors and covariance matrix using sum-of-weights
Minos(flag)	Controls if MINOS is run after HESSE
Save(flag)	if RooFitResult object is produced and returned
Strategy(flag)	Set Minuit strategy
ExternalConstraints(ArgSet)	Include given external constraints to likelihood
Extended(flag)	Add extended likelihood term
NumCPU(num)	Parallelize NLL calculation on num CPUs

Note: those options are defined under RooFit namespace

BROWSING FIT RESULTS WITH ROOFITRESULT

- As fits grow in complexity, number of output variables increases
  - Needs a better way to navigate the output that MINUIT screen dump, or keep the results for next step use.
- **RooFitResult** object holds a complete snapshot of the fit results
  - Constant parameters
  - Initial and final values of floating parameters
  - Global correlations & full correlation matrix

terminal output from example_08c.cc

```
RooFitResult: minimized FCN value: -8863.01,
              estimated distance to minimum: 1.57332e-06
              covariance matrix quality: Full, accurate covariance matrix
              Status : MINIMIZE=0 HESSE=0 MINOS=0

    Floating Parameter  InitialValue    FinalValue (+HiError,-LoError)    GblCorr.
  --------------------  ------------  ----------------------------------  --------
                    mu    5.0000e-01    5.0066e-01 (+9.97e-04,-9.97e-04)  <none>
                 sigma    1.0000e-01    9.9737e-02 (+7.09e-04,-7.02e-04)  <none>
```

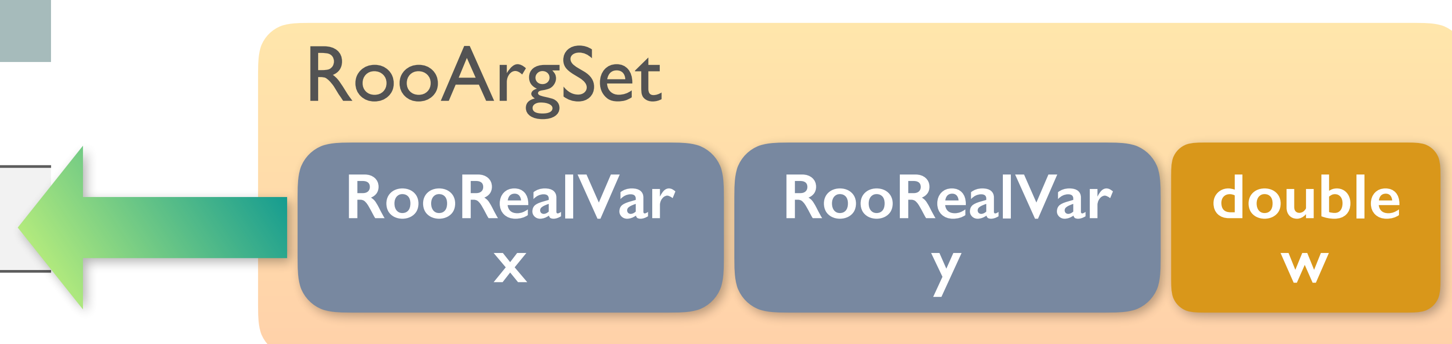
HANDLING DATASETS

- A <u>dataset</u> is a collection of data points in *N*-dimensional space
  - The dimensions can be either real or discrete.
  - Two dataset implementations based on a common abstract base class **`RooAbsData`**:

    `RooDataSet` – unbinned (*can be weighted or unweighted*)

    `RooDataHist` – binned
- Nearly all RooFit classes/functions (*in particular fitting*) take RooAbsData objects.
- Dataset structure:

index	x	y	w
0	1.0	3.2	1
1	5.2	7.5	1
2	0.5	6.2	1.2
3	3.3	1.4	2



CONSTRUCT A DATASET

- We already see a dataset can be easily generated using a given model by just calling "generate()".
- One can always build a dataset by adding each event:

example_11.cc

```
RooRealVar x("x","var x",0.,1.);
RooRealVar y("y","var y",-5.,+5.);
RooCategory c("c","charge");
c.defineType("Plus",+1);
c.defineType("Minus",-1);

RooDataSet data("data","data",RooArgSet(x,y,c)); ↶ empty dataset

TRandom3 rnd;
for (int i=0; i<1000; i++) {
    x.setVal(rnd.Uniform());
    y.setVal(rnd.Gaus());
    if (rnd.Uniform()<0.5) c.setLabel("Plus");
        else c.setLabel("Minus");
    data.add(RooArgSet(x,y,c)); ↶ add a new event as RooArgSet
}
data.Print("v");
```

```
DataStore data (data)
  Contains 1000 entries
  Observables:
    1)  x = 0.709477  L(0 - 1)  "var x"
    2)  y = 0.00310636  L(-5 - 5)  "var y"
    3)  c = Plus(idx = 1)  "charge"
```

IMPORTING DATASET FROM ROOT

- One can definitely convert a root tree (*and n-tuple*) to RooDataSet:
  - Branches with float point numbers or integer can be converted to RooRealVar;
  - Branches with integer or bool can be converted to RooCategory.
  - The name of branches are reserved: just pick up the branches you need!
- For example (converting from our early example data):

partial example_11.cc

```
{
    TFile *fin = new TFile("example_data.root");
    TNtupleD* nt = (TNtupleD *)fin->Get("nt");

    RooRealVar mass("mass","mass",0.,2.);
    RooDataSet data("data","data",nt,RooArgSet(mass));
    data.Print("v");
}
```

name of variable should match to the branch name

```
DataStore data (data)
  Contains 10000 entries
  Observables:
    1)  mass = 0.618399  L(0 - 2)  "mass"
```

IMPORTING DATASET FROM ROOT (CONT.)

➤ Binned data set can be also constructed from ordinary root histogram objects easily:

partial example_11.cc

```
TFile *fin = new TFile("example_data.root");
TH1D* hist = (TH1D *)fin->Get("hist");

RooRealVar mass("mass","mass",0.,2.);
RooDataHist bindata("bindata","bindata",RooArgList(mass),hist);
bindata.Print("v");
```

➤ Or, converting from the existing unbinned data:

partial example_11.cc

```
TFile *fin = new TFile("example_data.root");
TNtupleD* nt = (TNtupleD *)fin->Get("nt");

RooRealVar mass("mass","mass",0.,2.);
RooDataSet data("data","data",nt,RooArgSet(mass));

mass.setBins(50);
RooDataHist bindata2("bindata2","bindata2",RooArgList(mass),data);
bindata2.Print("v");
```

LET'S GO BACK TO OUR ORIGINAL FITS!

- Up to here we should be fully capable to perform simple fits as we discussed earlier with bare Minut.
- Here are the example to perform the standard ML fit:

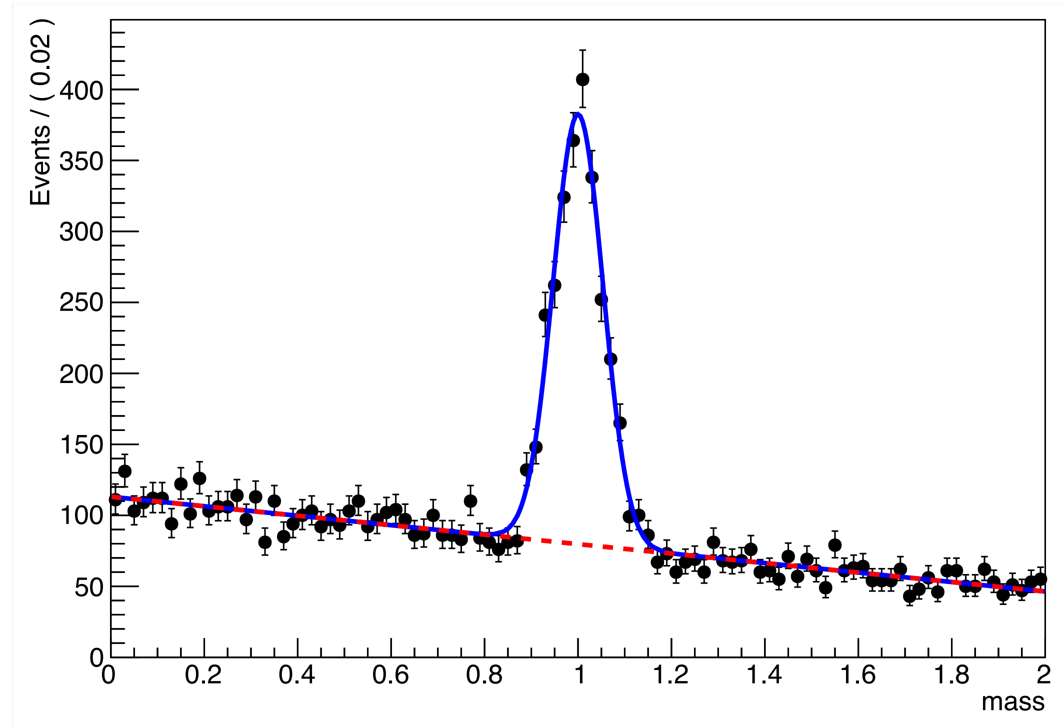
example_13.cc

```
TFile *fin = new TFile("example_data.root");
TNtupleD* nt = (TNtupleD *)fin->Get("nt");

RooRealVar mass("mass","mass",0.,2.);
RooDataSet data("data","data",nt,RooArgSet(mass));

RooRealVar mu("mu","mu",1.0,0.5,1.5);
RooRealVar sigma("sigma","sigma",0.05,0.001,0.15);
RooGaussian gaus("gaus","gaus",mass,mu,sigma);

RooRealVar slope("slope","slope",-0.3,-10.,10.);
RooPolynomial linear("linear","linear",mass,RooArgSet(slope));

RooRealVar frac("frac","frac",0.2,0.,1.);
RooAddPdf model("model","model",RooArgList(gaus,linear),RooArgList(frac));

model.fitTo(data,Minos(true));

RooPlot* frame = mass.frame();
data.plotOn(frame);
model.plotOn(frame);
model.plotOn(frame,Components(linear),LineStyle(7),LineColor(kRed));
frame->Draw();
```

Joint the Gaussian peak & linear background as before

UML EXAMPLE WITH ROOFIT

- The results should be consistent with what we did before with Minuit, except we are fitting the "fraction" of Gaussian peak:




```
FCN=5387.95 FROM MINOS     STATUS=SUCCESSFUL     56 CALLS 264 TOTAL
EDM=9.10578e-06    STRATEGY= 1      ERROR MATRIX ACCURATE
 EXT PARAMETER                PARABOLIC         MINOS ERRORS
 NO.   NAME      VALUE          ERROR       NEGATIVE      POSITIVE
  1  frac        2.02794e-01   5.47053e-03  -5.44941e-03   5.49189e-03
  2  mu          1.00033e+00   1.69593e-03  -1.69212e-03   1.70055e-03
  3  sigma       5.34462e-02   1.50505e-03  -1.48090e-03   1.53072e-03
  4  slope      -2.95052e-01   9.03755e-03  -8.88591e-03   9.19332e-03
```

Then how about the extended UML fit?

UML EXAMPLE WITH ROOFIT (CONT.)

➤ Extended UML fit is just a small change when constructing **RooAddPdf**:

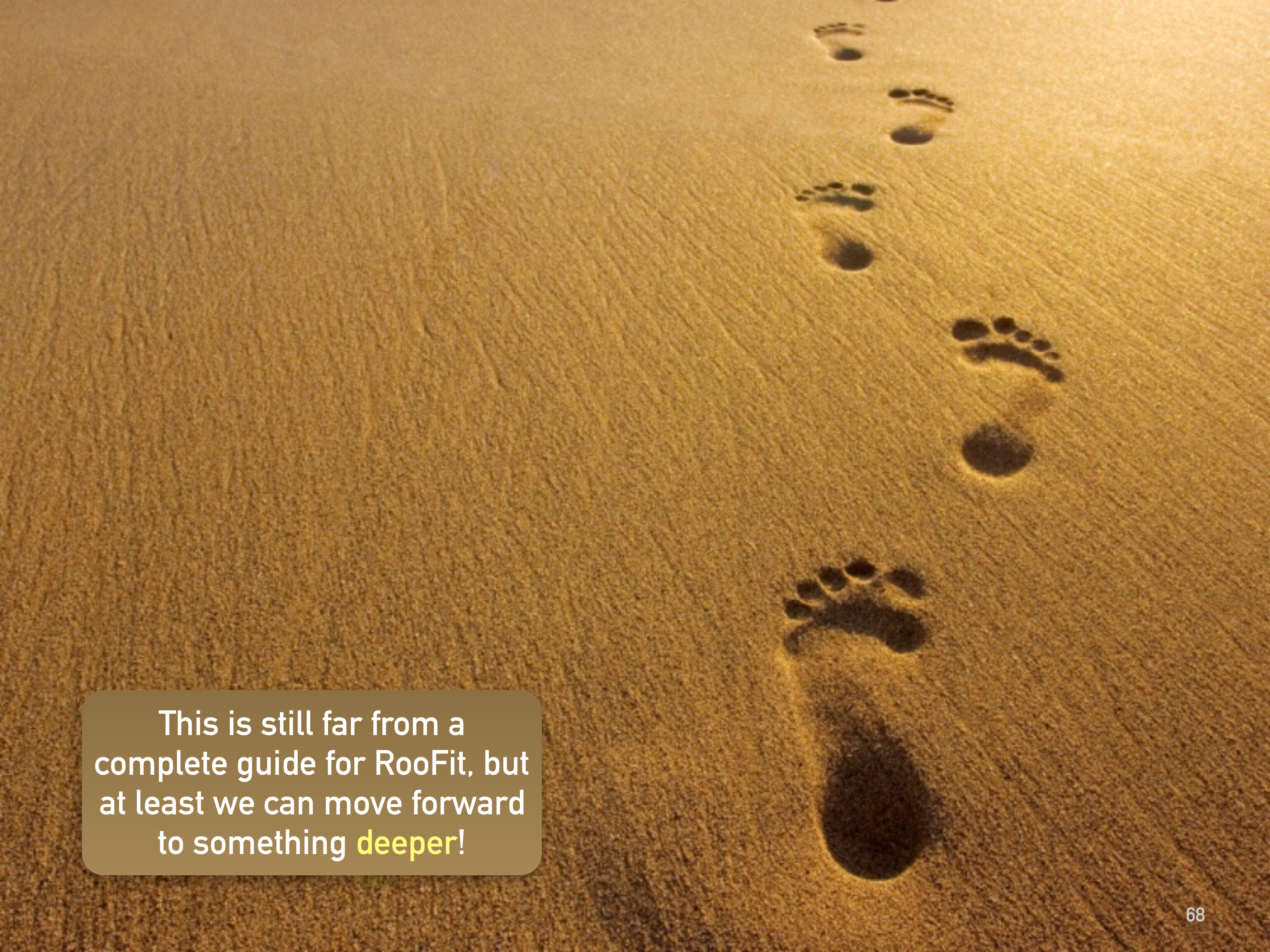
partial example_13a.cc

```
RooRealVar ns("ns","ns",2000,0.,20000.);
RooRealVar nb("nb","nb",8000,0.,20000.);
RooAddPdf model("model","model",RooArgList(gaus,linear),RooArgList(ns,nb));
```

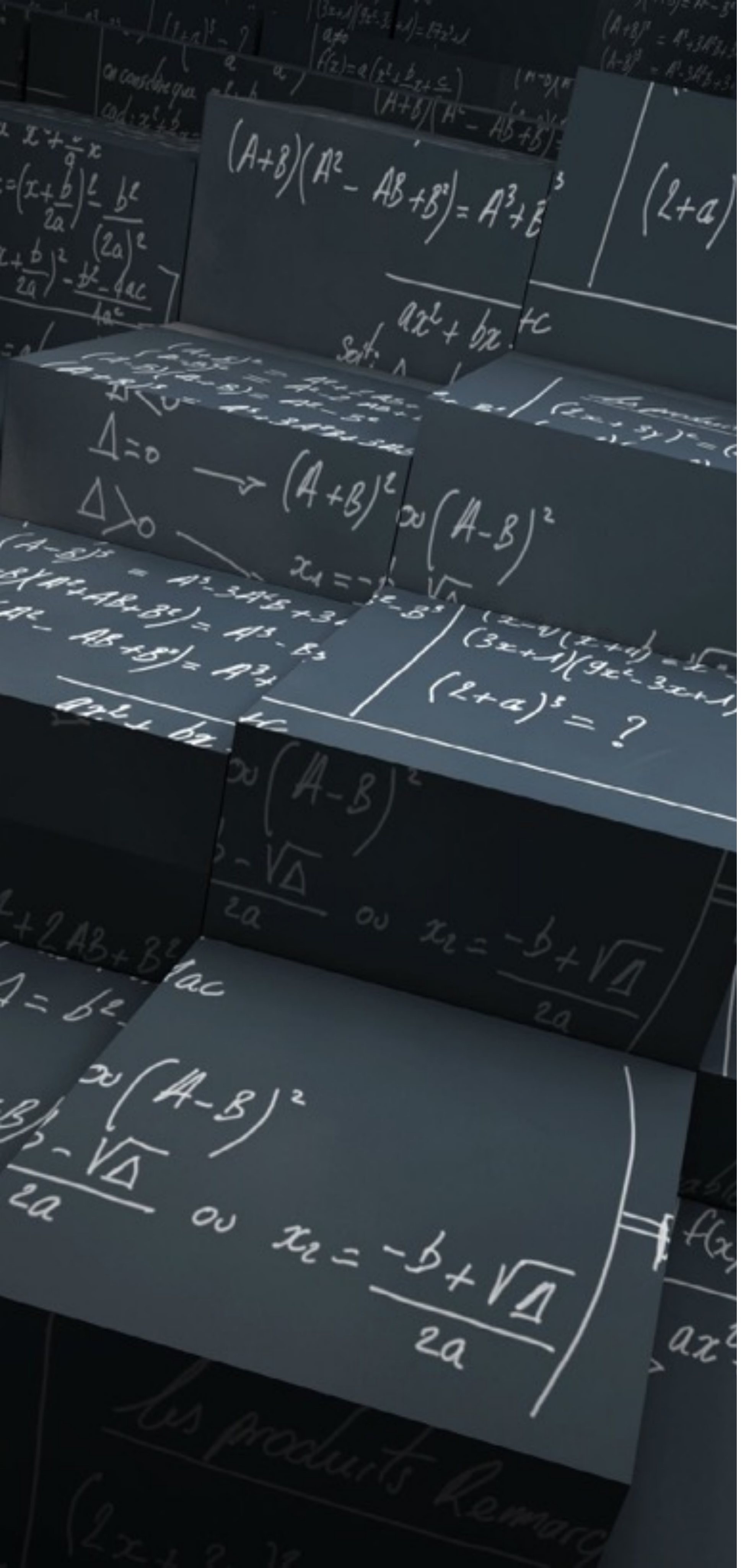
Now we need two "yields"

```
FCN=-76715.5 FROM MINOS     STATUS=SUCCESSFUL      81 CALLS 450 TOTAL
EDM=6.68556e-07     STRATEGY= 1        ERROR MATRIX ACCURATE
 EXT PARAMETER                        PARABOLIC          MINOS ERRORS
 NO.   NAME       VALUE                 ERROR       NEGATIVE       POSITIVE
  1   mu           1.00034e+00   1.69600e-03  -1.69522e-03   1.69742e-03
  2   nb           7.97206e+03   9.66839e+01  -9.63102e+01   9.70621e+01
  3   ns           2.02794e+03   5.83437e+01  -5.79784e+01   5.87150e+01
  4   sigma        5.34491e-02   1.50519e-03  -1.48389e-03   1.52773e-03
  5   slope       -2.95041e-01   9.03813e-03  -8.89745e-03   9.18170e-03
```

As good behaved as expected!



This is still far from a complete guide for RooFit, but at least we can move forward to something deeper!



SUMMARY

- In this inter-lecture we discussed how to use Minuit within ROOT, and how to perform some fits (least-square fit, maximum likelihood fit) with it.
- RooFit provided a framework to handle many nitty-gritty technical details of ML fitters. It should help when move toward a more complicated application.
- All we discussed here should cover the minimal technical needs of Minut and fitting, be prepared for the following lecture for more mathematics!