# Fitting: A guided overview

Angelo Di Canto

*Belle II Academy — March 25, 2021*

BROOKHAVEN
NATIONAL LABORATORY

# Basics about fitting
(for more see D. Tonelli's talk)

# What is fitting?

- Combining observed data *x* into a statistical model *p*(*x*|*m*) to infer the value of parameter *m* and its uncertainty

- Typically made by means of an estimator *e*(*x*) which is a function of the data *x*. Because the data are a random variable, so is the estimator *e*(*x*), which has its own probability distribution *p*[*e*(*x*)]

- For all practical purposes, the estimators that you will encounter in the great majority of analysis applications are the following:

  - Maximum-likelihood estimate is asymptotically ($N \to \infty$) consistent (unbiased), efficient (smallest possible variance), and normal (Gaussian distributed)

  - Least-squares estimate is asymptotically efficient for binned data, can be numerically more stable (particularly when the model depends linearly on *m*), and convenient when an assessment of goodness-of-fit is important

# Maximum likelihood

- The model $p(x|m)$ is the probability density function to observe a generic data point $x$, given the unobservable value of the parameters $m$

- The likelihood is computed by taking the actual observed data points $x_i$ and evaluate

$$L(m) = \prod_{i}^{\text{events}} p(x_i|m)$$

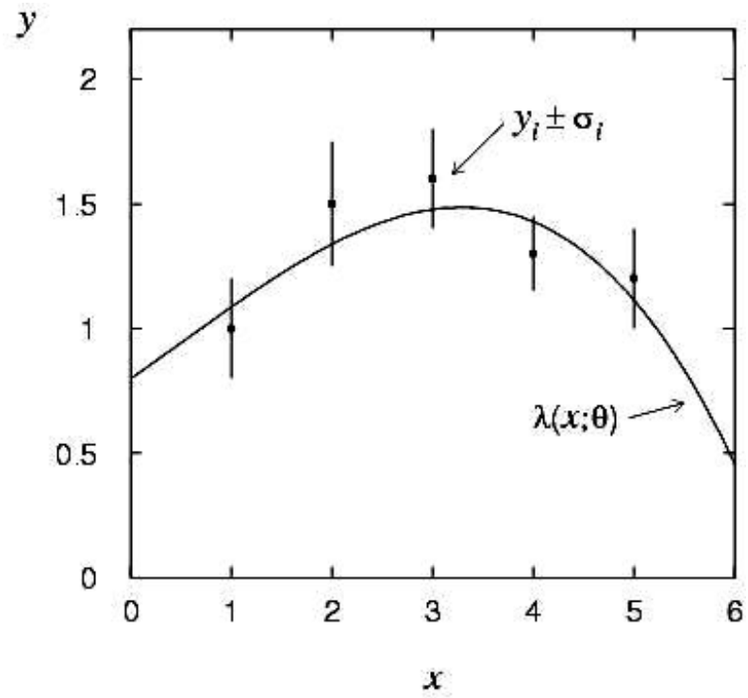
- The likelihood expresses the probability of observing data $x$ for different values of the parameter $m$ (not the probability that m has some value given the data)

- Given the data, the parameter values $m_{\text{low}}$ that decrease $L(m)$ are disfavored as it would be unlikely for nature to generate that set of observed data, had the true value of m been $m_{\text{low}}$. Conversely, values $m_{\text{high}}$ that increase $L(m)$ are favored

    - The value of $m$ that maximizes the likelihood is not the “most likely value of $m$”, it is the value of $m$ that makes your data most likely

# Least-squares from the maximum-likelihood

Assume to have *N* independent observations $y_1,\ldots,y_N$ that fluctuate following Gaussian distributions of known variance $V[y_i] = \sigma_i^2$ .




around their known expected values

$$E[y_i] = \lambda(x_i;\theta) \ .$$

that are functions of a known variable $x_i$ and unknown parameter $\theta$.

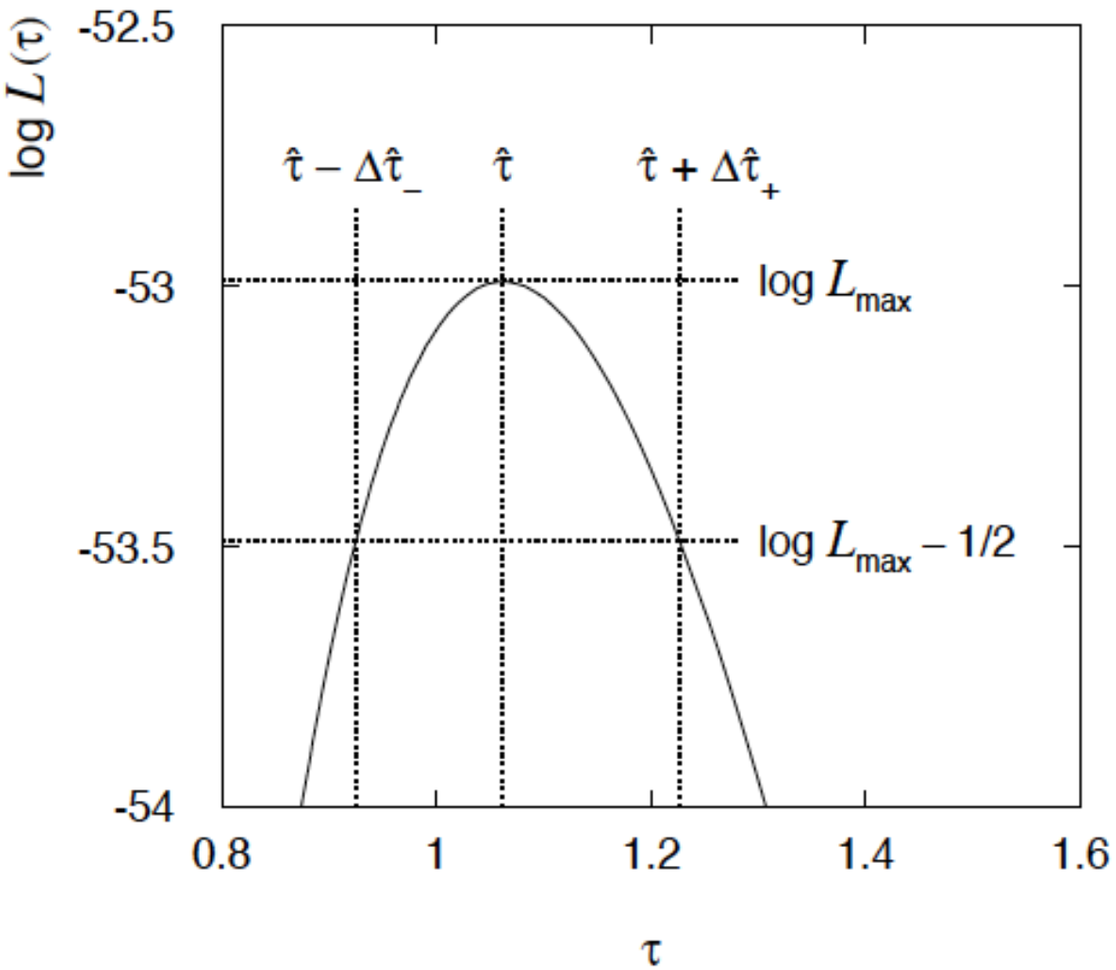
The likelihood function is $L(\theta) = \prod_{i=1}^{N} f(y_i;\theta) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left[-\frac{(y_i - \lambda(x_i;\theta))^2}{2\sigma_i^2}\right]$

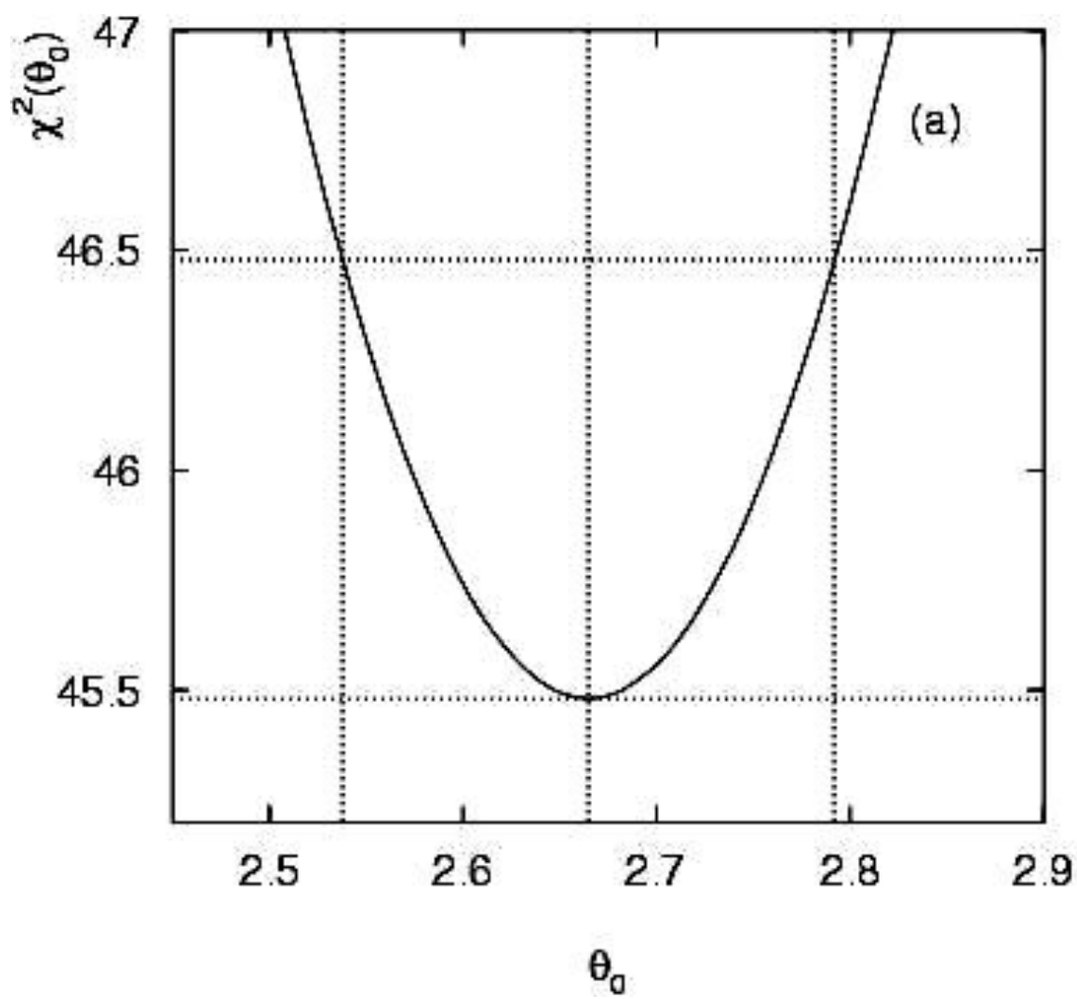
The logarithm is $\ln L(\theta) = -\frac{1}{2}\sum_{i=1}^{N} \frac{(y_i - \lambda(x_i;\theta))^2}{\sigma_i^2} +$ terms not depending on $\theta$

Hence, maximizing the likelihood is equivalent to minimizing the least-squares

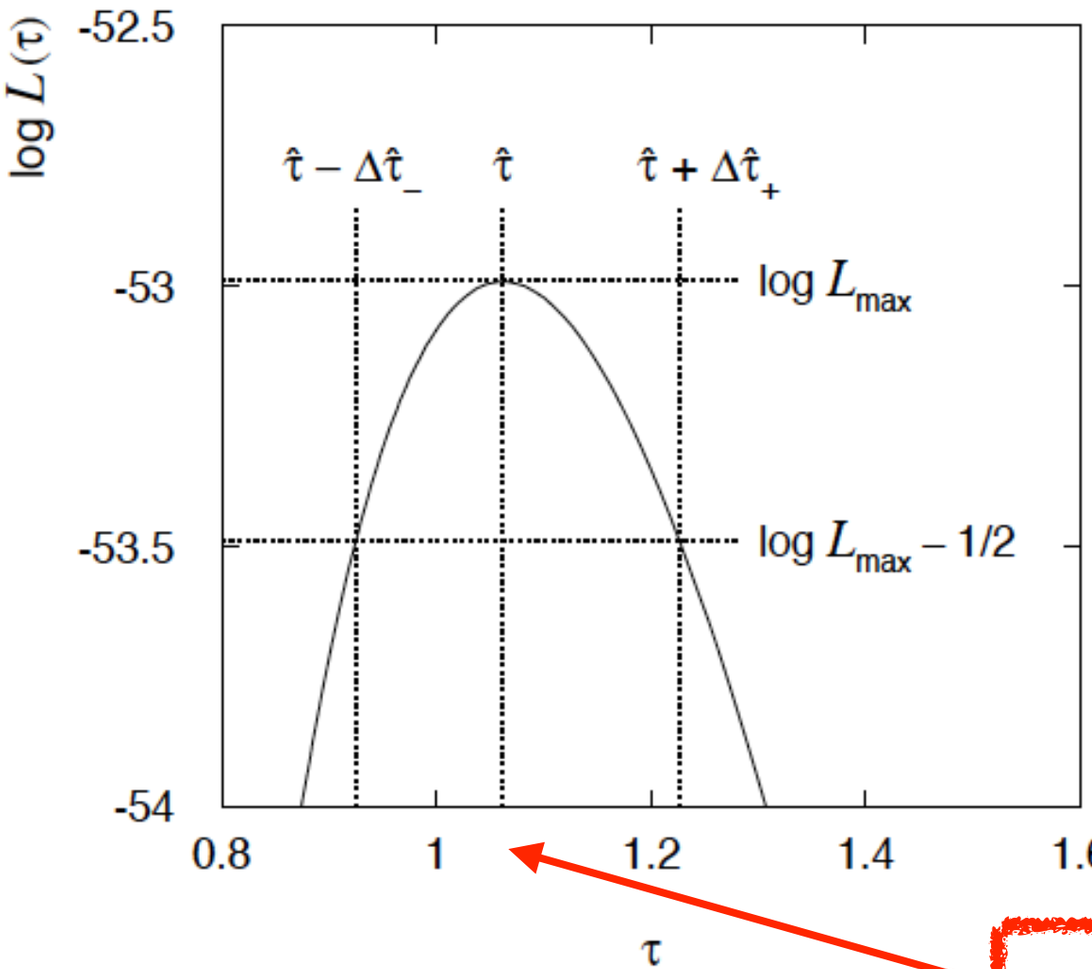
$$\chi^2(\theta) = -2\ln L(\theta) = \sum_{i=1}^{N} \frac{(y_i - \lambda(x_i;\theta))^2}{\sigma_i^2}$$

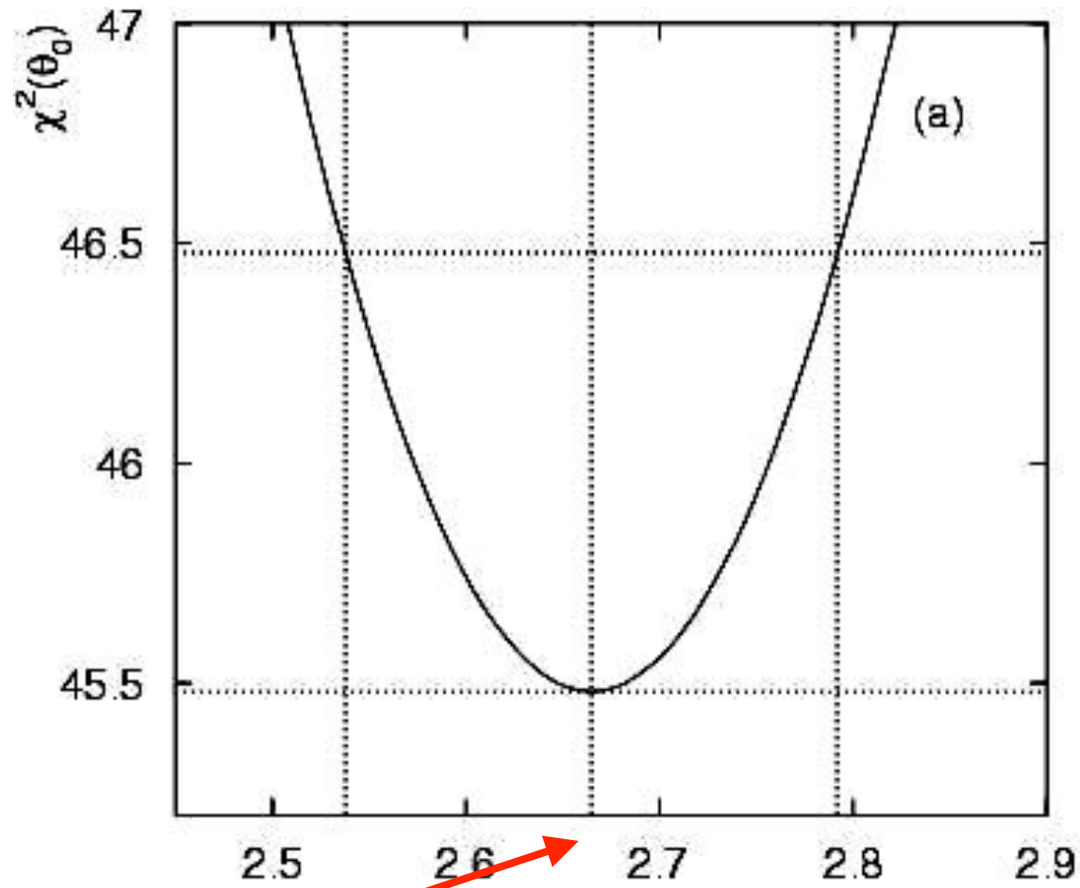
# Maximum-likelihood/least-squares on a plot



# Maximum-likelihood/least-squares on a plot

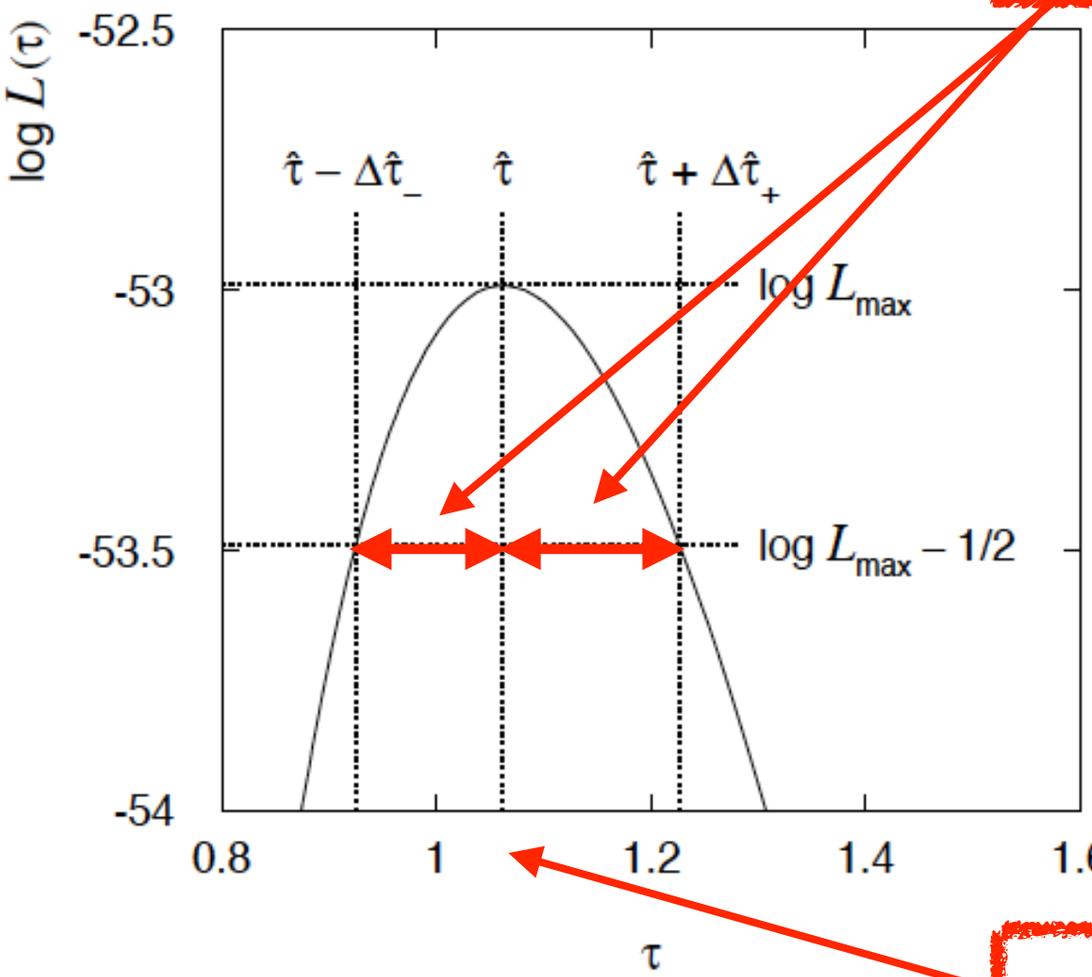


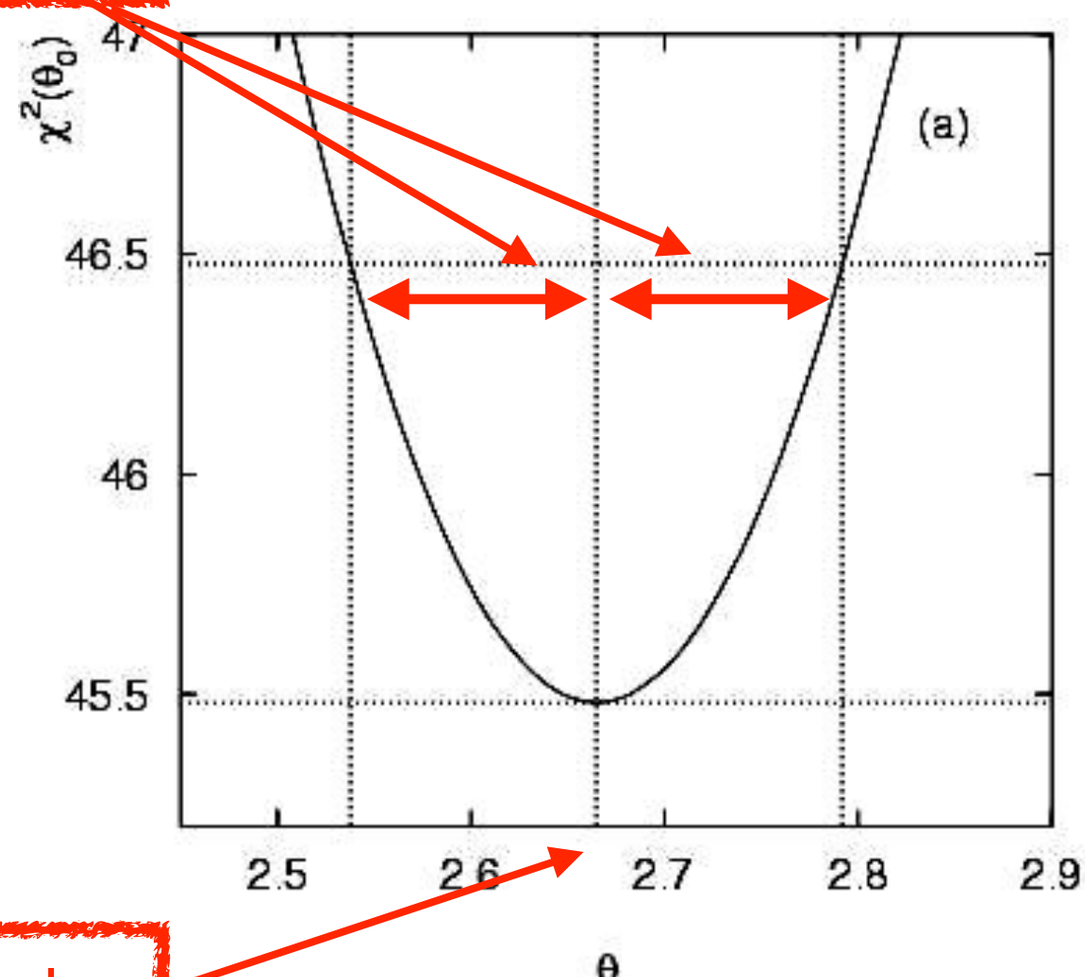




central value

# Maximum-likelihood/least-squares on a plot

Uncertainty



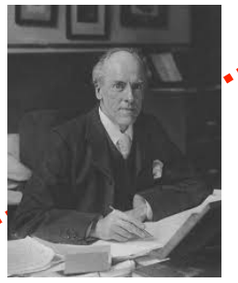

central value

$$\log L(\hat{m} \pm \hat{\sigma}) \approx \log L_{\max} - \frac{1}{2}$$

$$\chi^2(\hat{m} \pm \hat{\sigma}) \approx \chi^2_{\min} + 1$$

# Fitting roadmap by D. Tonelli

Assume model $p(x|m)$



I absolutely need GOF or have so many data that unbinned ~ binned

Least-squares fit

**Lots of simulation**

I need most precise estimate and/ or have a small data sample

Maximum likelihood fit to unbinned data

**Lots of simulation**

Check estimator properties

# Which fitting framework shall I use?

- Any that you understand and that best suits your needs
- The most used/recommended in HEP is RooFit

https://root.cern/manual/roofit/

- However, as most other fitting frameworks, RooFit relies on Minuit to perform the minimization and estimate the uncertainties
- To understand your fitting framework you most likely want to understand Minuit first

# What is Minuit?

- A standalone package to find/calculate numerically
  - The (local) minimum of any arbitrary function $F(\boldsymbol{p})$, where $\boldsymbol{p}$ is a set of parameters (typically least-squares or negative log-likelihood)
  - The covariance matrix of these parameters (at the minimum)

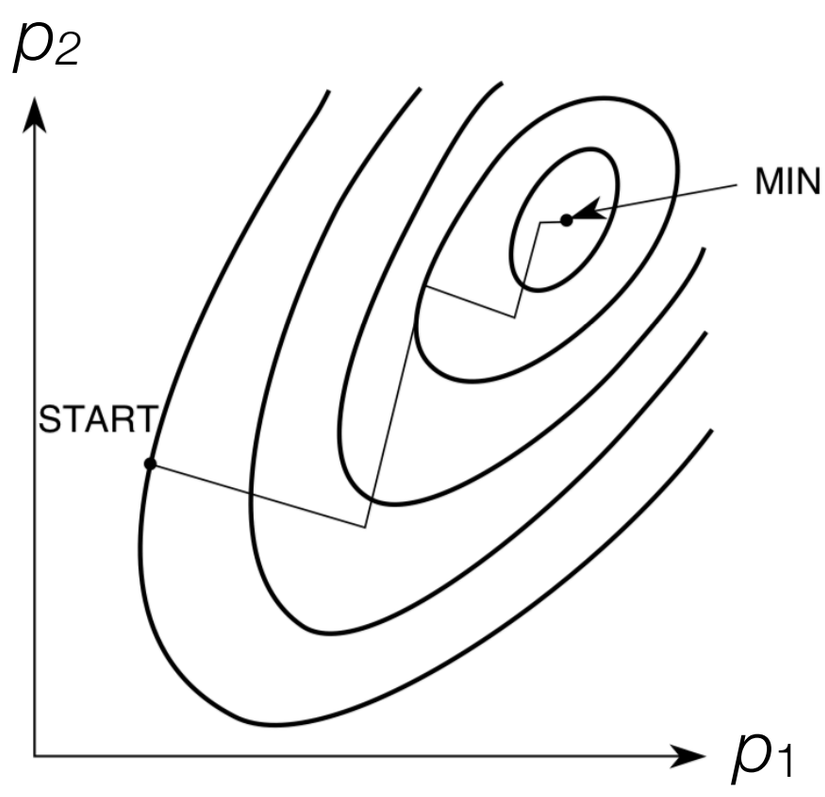
- Minuit was originally written in fortran by F. James, and then adapted to C++ within ROOT by R. Brun
- F. James and M. Winkler have also re-designed and re-implemented the algorithm in C++ (Minuit2)
- There's also a python wrapper called `iminuit`

# What exactly minimization means

- The function to minimize, *F*, does not need to be known analytically. It is sufficient to know its value *F*(***p***), at any point ***p***

- Minuit looks for a local minimum: *i.e.*, the point $\hat{\boldsymbol{p}}$ where $F(\hat{\boldsymbol{p}}) < F(\boldsymbol{p})$ for any ***p*** in some neighborhood around $\hat{\boldsymbol{p}}$ (different starting points may result in different minima)

- The general strategy for finding a local minimum is simply to vary ***p*** by small steps, in a direction which causes *F* to decrease, until one finds the point $\hat{\boldsymbol{p}}$ from which *F* increases in all allowed directions




- Although not needed, if the numerical values of the derivative $\partial F(\boldsymbol{p})/\partial \boldsymbol{p}$ at any point ***p*** is known, they can be used to help in the minimization

# What Minuit does not

- All the rest that has to do with fitting:
  - Data handling
  - Graphics (data visualization, fit projections, *etc*…)
  - Validation with pseudoexperiments
  - *etc*…
- (These instead are typically available in fitting frameworks such as RooFit)

# Minuit main algorithms

- `MIGRAD` — Performs a local minimization of the `FCN` using a variable-step method based on the estimated direction of the gradient. The minimization produces as a by-product also the error matrix of the parameters

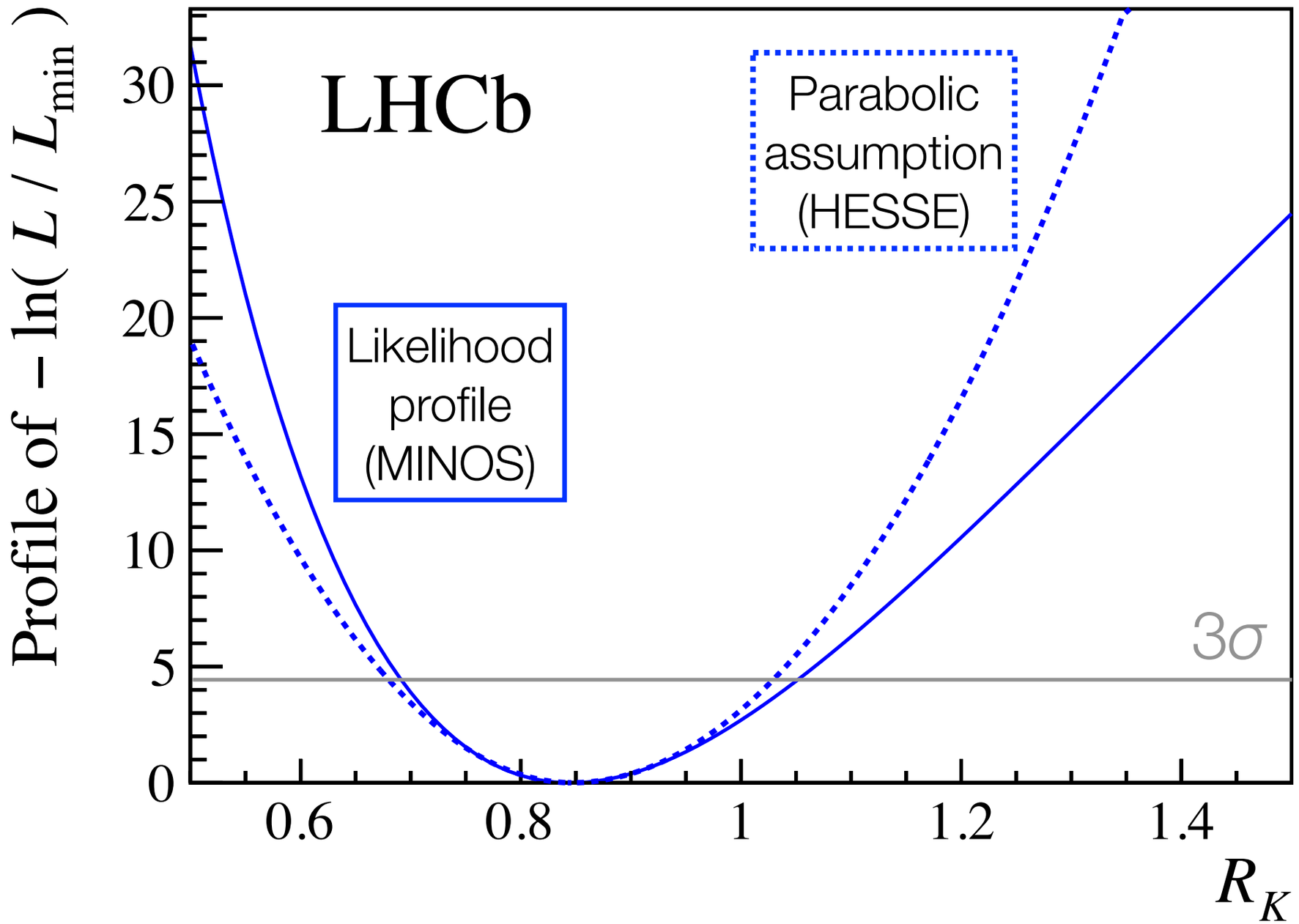
- `HESSE` — Calculates the full second-derivative matrix of the `FCN` using a finite difference method. Used to improve the estimation of the parabolic errors obtained by `MIGRAD`

- `MINOS` — Performs a scan of the `FCN`, profiled in each given dimension (*i.e.,* by minimizing all other parameters at each scan point), around the local minimum to estimate asymmetric errors

    - `CONTOUR` — Performs a scan of the `FCN`, profiled in the given two dimensions, around the local minimum to estimate border (contour) of 2D confidence-level intervals

# HESSE vs MINOS

[PRL 122 (2019) 191801]



# Some practical examples

# Instructions

- Get the fitting examples from b2-fitting-training

  ```
  git clone ssh://git@stash.desy.de:7999/b2t/b2-fitting-training.git
  cd b2-fitting-training/
  ```

- Sphinx documentation

- Today we concentrate on the examples based on Minuit (which are written in C++ and require to be compiled)

  ```
  cd minuit
  ```

# Instructions (if you are not using KEKCC)

- To compile you just need CMake, ROOT and TCLAP. If they are installed in your system, compile all examples with

```
mkdir build; cd build
cmake ../
make
```

- If TCLAP is not installed in your system, download it and run cmake with

```
cmake -D TCLAP_PATH=path_to_tclap_dir ../
```

- If CMake or ROOT are not installed… then work from KEKCC

# Instructions (for KEKCC)

- On KEKCC you can get CMake and ROOT by setting up any recent `basf2` release

```
source /cvmfs/belle.cern.ch/tools/b2setup release-04-01-04
```

- Then follow the instructions from the previous slide but specify gcc/g++ as compilers when running the cmake command with

```
mkdir build; cd build
cmake -D CMAKE_C_COMPILER=`which gcc` \
            -D CMAKE_CXX_COMPILER=`which g++` \
               -D TCLAP_PATH=path_to_tclap_dir ../
make
```

# How to run the examples

- The compiled executables are in the bin directory

  `ls ../bin`

- To understand how to use any of them, run with the `–h` argument, *e.g.* do

  ```
  cd ..
  ./bin/simple-1d-fit –h
  ```

- To run do

  ```
  mkdir output
  ./bin/simple-1d-fit –p –o output
  ```

# Example 1: simple 1D unbinned likelihood fit

# Simple 1D fit

- Unbinned likelihood fit to the beam-constrained mass distribution of $B^0 \to K^*\gamma$ decay candidates reconstructed in simulation, to determine the fraction of signal decays

- The input data is provided by the ROOT ntuple `BtoKstG` contained in the file `example-data/fitme.root`. The branch of the tree corresponding to the beam-constrained mass is `B0_mbc`

- The example code using the ROOT's `TFitter` interface to Minuit (but the basic concepts are independent of the interface)

# Design a fitter based on Minuit

- Conceptual steps:
  - Prepare the data to fit to
  - Write the function to minimize (*i.e.*, choose model and estimator)
  - Setup Minuit
    - Define the parameters of the fit, their starting values and their allowed ranges
    - Specify the sequence of algorithms to use for minimization and estimation of the covariance matrix
    - Configure each algorithm
- Access fit results
- Plot the results for graphical visualization
- Prepare tools for validation of the fitter (*e.g.*, generation of pseudoexperiments)

# The function to minimize: `FCN`

$$-2\log L(\vec{\theta}) = -2\sum_{i\in\text{data}} \log\ \text{pdf}(m_i|\vec{\theta})$$

```
49  // Computation of -2*log(likelihood)
50  void logLfun(int &/*npar*/, double * /*gin*/, double &result, double *pars, int /* flag */)
51  {
52     result = 0.;
53     for (auto m : data) {
54        double prob = fit_pdf(&m,pars);
55        if (prob<=0.) prob = 1e-300;
56        result -= 2.*log(prob);
57     }
58  }
```

# Fit model

- Assuming two components: signal described by a Crystal Ball with *n* fixed to 15, and background described by Argus

$$\mathrm{pdf}_{\mathrm{sgn}}(m|\mu,\sigma,\alpha) \propto \begin{cases} e^{-\frac{1}{2}\left(\frac{m-\mu}{\sigma}\right)^2} & \text{if} \quad \frac{m-\mu}{\sigma} > \alpha \\ \left(\frac{n}{|\alpha|} - |\alpha| - \frac{m-\mu}{\sigma}\right)^{-n} & \text{if} \quad \frac{m-\mu}{\sigma} \leq -\alpha \end{cases}$$

$$\mathrm{pdf}_{\mathrm{bkg}}(m|m_0,c) \propto \begin{cases} \frac{m}{m_0^3}\sqrt{1-\frac{m^2}{m_0^2}}\, e^{-\frac{1}{2}c^2\left(1-\frac{m^2}{m_0^2}\right)} & \text{if} \quad m \leq m_0 \\ 0 & \text{if} \quad m > m_0 \end{cases}$$

- The total PDF is the sum weighted by the signal fraction

$$\mathrm{pdf}(m|\mu,\sigma,\alpha,m_0,c) = f_{\mathrm{sgn}}\,\mathrm{pdf}_{\mathrm{sgn}}(m|\mu,\sigma,\alpha) + (1-f_{\mathrm{sgn}})\,\mathrm{pdf}_{\mathrm{bkg}}(m|m_0,c)$$

# Setup Minuit (using ROOT's `TFitter` interface)

```
93    // Configure MINUIT
94    TFitter *fitter = new TFitter(npars);
95    fitter->SetFCN(logLfun);
96
97    double printlevel(args.prlevel);
98    fitter->ExecuteCommand("SET PRI",&printlevel,1);
99    if (printlevel<0) fitter->ExecuteCommand("SET NOW",&printlevel,0);
100
101   double strategy(2.);
102   fitter->ExecuteCommand("SET STRAT",&strategy,1);
103
104   double errdef(1.);
105   fitter->ExecuteCommand("SET ERR",&errdef,1);
106
107   double eps_machine(std::numeric_limits<double>::epsilon());
108   fitter->ExecuteCommand("SET EPS",&eps_machine,1);
109
110   // Define and initialize parameters
111   fitter->SetParameter(0,      "f_{sig}",  0.7, 1e-3, 0.0, 1.0);
112   fitter->SetParameter(1,     "#mu_{CB}", 5.28, 1e-3, 5.2, 5.3);
113   fitter->SetParameter(2,  "#sigma_{CB}", 3e-3, 1e-4, 0.0, 1.5e-2);
114   fitter->SetParameter(3,  "#alpha_{CB}",  1.3, 1e-3, 0.1, 5.0);
115   fitter->SetParameter(4,       "n_{CB}",  15., 1e-3, 0.0, 0.0);
116   fitter->FixParameter(4);
117   fitter->SetParameter(5,   "m_{cutoff}", 5.29, 1e-3, 5.2, 5.3);
118   fitter->SetParameter(6,"c_{curvature}", -20.,  1.0,-80.,-1.0);
119
120   // Run minimization and compute uncertainties
121   double arglist[] = {5000., 1.};
122   fitter->ExecuteCommand("MIGRAD",arglist,2);
123   fitter->ExecuteCommand("HESSE",arglist,2);
```

- The arguments of `ExecuteCommand` are the same as those used in fortran

# Setup Minuit

- Set minimization strategy to 2 for reliable results

**SET STRategy** `<level>`

Sets the strategy to be used in calculating first and second derivatives and in certain minimization methods. In general, low values of `<level>` mean fewer function calls and high values mean more reliable minimization. Currently allowed values are 0, 1 (default), and 2.

- Set errors' definition according to your use-case (1D 68% CL intervals corresponds to $\Delta\chi$=1 or $\Delta\log L$ = 0.5)

**SET ERRordef** `<up>`

Sets the value of `UP` (default value= 1.), defining parameter errors. Minuit defines parameter errors as the change in parameter value required to change the function value by `UP`. Normally, for chisquared fits `UP=1`, and for negative log likelihood, `UP=0.5`.

# Setup Minuit

- Define the fit parameters

◆ SetParameter()

```
Int_t TFitter::SetParameter ( Int_t          ipar,
                              const char *   parname,
                              Double_t       value,
                              Double_t       verr,
                              Double_t       vlow,
                              Double_t       vhigh
                            )                                   virtual
```

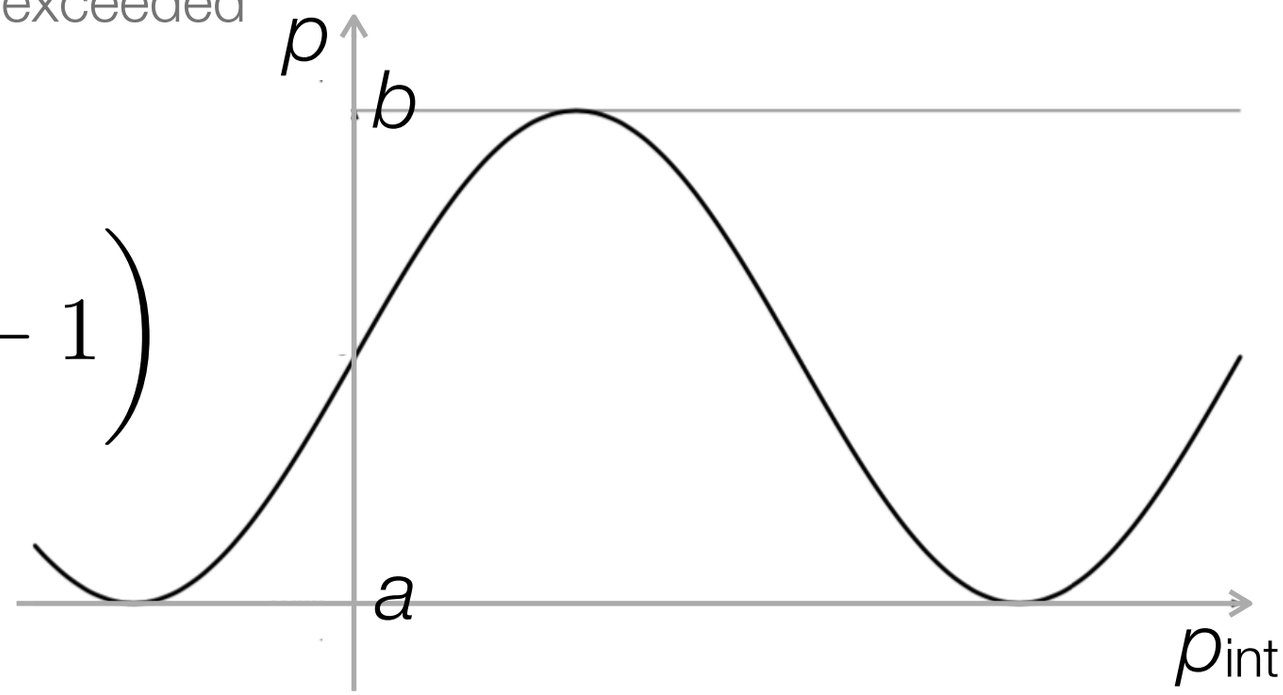
set initial values for a parameter

- ipar : parameter number
- parname : parameter name
- value : initial parameter value
- verr : initial error for this parameter ← initial step size (if set to 0, parameter remain fixed)
- vlow : lower value for the parameter } boundaries (none, if both set to 0)
- vhigh : upper value for the parameter

# Parameter boundaries

- When boundaries on a parameter are specified, Minuit internally converts the parameter such that the boundaries cannot be exceeded

$$p_{\text{int}} = \arcsin\left(2\,\frac{p-a}{b-a} - 1\right)$$




- Boundaries should be avoided: they complicate the problem (since the above transformation is non-linear) and, more importantly, they may affect the estimation of the error matrix by `HESSE` (when a parameter gets close to the limit, the error matrix becomes singular)

- Workaround when that is not possible: (1) find the minimum with boundaries, (2) release the boundaries, (3) rerun `MIGRAD` and `HESSE` to confirm to be in a minimum and compute the uncertainties

# Run the minimization

- Call MIGRAD

**MIGrad** [maxcalls] [tolerance]

Causes minimization of the function by the method of Migrad, the most efficient and complete single method, recommended for general functions (see also MINImize). The minimization produces as a by-product the error matrix of the parameters, which is usually reliable unless warning messages are produced. The optional argument [maxcalls] specifies the (approximate) maximum number of function calls after which the calculation will be stopped even if it has not yet converged. The optional argument [tolerance] specifies required tolerance on the function value at the minimum. The default tolerance is 0.1, and the minimization will stop when the estimated vertical distance to the minimum (EDM) is less than 0.001*[tolerance]*UP (see SET ERR).

- One should use at least HESSE (same arguments as MIGRAD) after MIGRAD to obtain reliable errors for a given fit result. MINOS will give the best estimate of the errors, but may be computationally expensive (particularly for large numbers of free parameters)

```
Read 7205 data candidates.
**********
**    1 **SET PRI           1
**********
**********
**    2 **SET STRAT           2
**********
NOW USING STRATEGY  2: MAKE SURE MINIMUM TRUE, ERRORS CORRECT
**********
**    3 **SET ERR           1
**********
**********
**    4 **SET EPS    2.22e-16
**********
FLOATING-POINT NUMBERS ASSUMED ACCURATE TO 2.22045e-16
PARAMETER DEFINITIONS:
   NO.   NAME         VALUE      STEP SIZE      LIMITS
    1 f_{sig}      7.00000e-01  1.00000e-03    0.00000e+00  1.00000e+00
    2 #mu_{CB}     5.28000e+00  1.00000e-03    5.20000e+00  5.30000e+00
    3 #sigma_{CB}   3.00000e-03  1.00000e-04    0.00000e+00  1.50000e-02
    4 #alpha_{CB}   1.30000e+00  1.00000e-03    1.00000e-01  5.00000e+00
    5 n_{CB}       1.50000e+01  1.00000e-03     no limits
**********
**    5 **FIX           5
**********
PARAMETER DEFINITIONS:
   NO.   NAME         VALUE      STEP SIZE      LIMITS
    6 m_{cutoff}   5.29000e+00  1.00000e-03    5.20000e+00  5.30000e+00
    7 c_{curvature}  -2.00000e+01  1.00000e+00   -8.00000e+01 -1.00000e+00
**********
**    6 **MIGRAD        5000           1
**********
FIRST CALL TO USER FUNCTION AT NEW START POINT, WITH IFLAG=4.
START MIGRAD MINIMIZATION.  STRATEGY  2.  CONVERGENCE WHEN EDM .LT. 1.00e-03
MINUIT WARNING IN HESSE
```

```
MIGRAD MINIMIZATION HAS CONVERGED.
MIGRAD WILL VERIFY CONVERGENCE AND ERROR MATRIX.
COVARIANCE MATRIX CALCULATED SUCCESSFULLY
FCN=-42065.7 FROM MIGRAD    STATUS=CONVERGED      521 CALLS         522 TOTAL
                     EDM=3.25071e-05    STRATEGY= 2      ERROR MATRIX ACCURATE
 EXT PARAMETER                                   STEP         FIRST
 NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
  1  f_{sig}      5.43992e-01   1.09225e-02   7.25749e-04  -4.63661e-02
  2  #mu_{CB}     5.27958e+00   8.80003e-05   8.51942e-05  -1.06408e+00
  3  #sigma_{CB}   3.32971e-03   7.43293e-05   4.40399e-04   1.81416e-02
  4  #alpha_{CB}   1.08400e+00   7.93500e-02   1.23038e-03  -1.53757e-01
  5  n_{CB}       1.50000e+01     fixed
  6  m_{cutoff}   5.28980e+00   1.44704e-04   2.31412e-04   1.31000e+00
  7  c_{curvature}  -4.41732e+01   3.29219e+00   2.97963e-03  -1.14514e-02
EXTERNAL ERROR MATRIX.    NDIM=  25    NPAR=  6    ERR DEF=1
 1.193e-04  1.673e-07  1.000e-07 -4.520e-04 -3.948e-08  2.416e-02
 1.673e-07  7.744e-09 -3.093e-09 -3.991e-06  6.639e-11  4.353e-05
 1.000e-07 -3.093e-09  5.525e-09  2.665e-06 -1.821e-09  3.362e-05
-4.520e-04 -3.991e-06  2.665e-06  6.300e-03 -1.087e-06 -1.160e-01
-3.948e-08  6.639e-11 -1.821e-09 -1.087e-06  2.094e-08  3.655e-05
 2.416e-02  4.353e-05  3.362e-05 -1.160e-01  3.655e-05  1.086e+01
PARAMETER  CORRELATION COEFFICIENTS
       NO.  GLOBAL      1      2      3      4      6      7
        1  0.75053   1.000  0.174  0.123 -0.521 -0.025  0.671
        2  0.62698   0.174  1.000 -0.473 -0.571  0.005  0.150
        3  0.67554   0.123 -0.473  1.000  0.452 -0.169  0.137
        4  0.79403  -0.521 -0.571  0.452  1.000 -0.095 -0.443
        6  0.24386  -0.025  0.005 -0.169 -0.095  1.000  0.077
        7  0.70424   0.671  0.150  0.137 -0.443  0.077  1.000
```

```
**********
**    7 **HESSE         5000                1
**********
COVARIANCE MATRIX CALCULATED SUCCESSFULLY
FCN=-42065.7 FROM HESSE      STATUS=OK                 40 CALLS         562 TOTAL
                    EDM=3.25445e-05     STRATEGY= 2      ERROR MATRIX ACCURATE
 EXT PARAMETER                                          INTERNAL      INTERNAL
 NO.   NAME       VALUE            ERROR          STEP SIZE       VALUE
  1  f_{sig}      5.43992e-01   1.09214e-02   1.45150e-04   8.80987e-02
  2  #mu_{CB}     5.27958e+00   8.80710e-05   1.70388e-05   6.33034e-01
  3  #sigma_{CB}   3.32971e-03   7.44537e-05   1.76159e-05  -5.89613e-01
  4  #alpha_{CB}   1.08400e+00   7.93465e-02   2.46075e-04  -1.32078e+01
  5  n_{CB}       1.50000e+01     fixed
  6  m_{cutoff}   5.28980e+00   1.44897e-04   9.25649e-06   9.20754e-01
  7  c_{curvature}  -4.41732e+01   3.29116e+00   5.95925e-04  -9.31271e-02
EXTERNAL ERROR MATRIX.    NDIM=  25    NPAR=  6    ERR DEF=1
 1.193e-04  1.674e-07  1.002e-07 -4.512e-04 -4.214e-08  2.415e-02
 1.674e-07  7.757e-09 -3.107e-09 -4.000e-06  7.183e-11  4.352e-05
 1.002e-07 -3.107e-09  5.544e-09  2.678e-06 -1.857e-09  3.368e-05
-4.512e-04 -4.000e-06  2.678e-06  6.299e-03 -1.095e-06 -1.157e-01
-4.214e-08  7.183e-11 -1.857e-09 -1.095e-06  2.100e-08  3.583e-05
 2.415e-02  4.352e-05  3.368e-05 -1.157e-01  3.583e-05  1.086e+01
PARAMETER  CORRELATION COEFFICIENTS
       NO.   GLOBAL      1      2      3      4      6      7
        1  0.75047   1.000  0.174  0.123 -0.520 -0.027  0.671
        2  0.62779   0.174  1.000 -0.474 -0.572  0.006  0.150
        3  0.67689   0.123 -0.474  1.000  0.453 -0.172  0.137
        4  0.79433  -0.520 -0.572  0.453  1.000 -0.095 -0.443
        6  0.24599  -0.027  0.006 -0.172 -0.095  1.000  0.075
        7  0.70402   0.671  0.150  0.137 -0.443  0.075  1.000
```

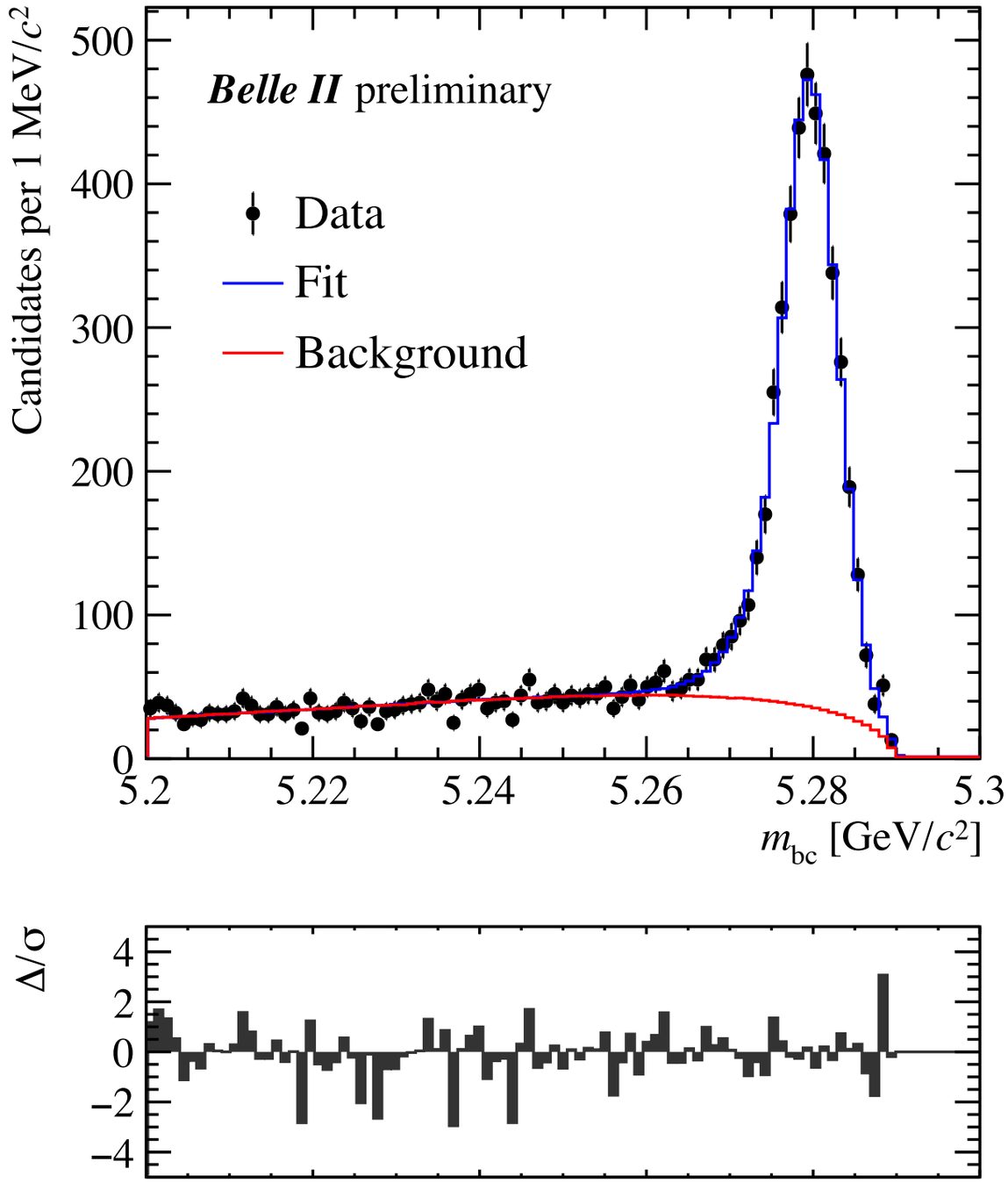
# Status of covariance matrix from the code

```
125     // Access status of the covariance matrix
126     TMinuit *minuit = fitter->GetMinuit();
127     double fmin, fedm; int npari, nparx, istat;
128     minuit->mnstat(fmin,fedm,errdef,npari,nparx,istat);
129     if (istat!=3) {
130        // istat is a status integer indicating how good is the covariance matrix:
131        //  0= not calculated at all
132        //  1= approximation only, not accurate
133        //  2= full matrix, but forced positive-definite
134        //  3= full accurate covariance matrix
135        std::cout << "Covariance matrix status = " << istat << std::endl;
136        return -2;
137     }
```

# Fit projections

```
------------------------
Set Belle II Style
------------------------
Plotting fit projection...
Generated 7205000 toy candidates.
Generated 3285534 toy candidates.
chi2/ndf (prob) = 88.9847/89 (0.480519)
```

This is the $\chi^2$ resulting from the comparison of the binned data with the fit projection $\Longrightarrow$ it does not quantify the goodness of fit!



# Goodness-of-fit for unbinned likelihood fits

- Goodness-of-fit is built-in in least-squares estimates. Can we devise a solid goodness-of-fit determination for unbinned MLE too?

- Some (*e.g.*, G. Cowan book) suggest to use the distribution of the value of the likelihood at its maximum as a distribution from which to extract a *p*-value. It is easy to demonstrate that such approach is flawed, for example see arXiv:physics/0310167

- Others (e.g., arXiv:006.3019) have cooked up various ad-hoc methods and claim they achieve the desired goal, but no general demonstration of their success and properties is given, so no guarantee exists that they'll work in general problems

- To date, no widely accepted method for evaluating goodness-of-fit in unbinned fits exists

  - Approximated goodness-of-fit measures based on binning the unbinned data offer a semiqualitative indication of the compatibility with the model and are usually reported when goodness-of-fit is important in unbinned fits

# Goodness-of-fit for least-squares fits

- If the model is correct, the value of the least-squares at the minimum LS($\hat{\theta}$) is distributed like a $\chi^2$ with ndf equal to the number of points minus the number of free parameters

- Since $E(\chi^2)$ = ndf, some usually quote $\chi^2$/ndf as measurement of the goodness-of-fit. That's flawed, you should instead use the probability (computed by `TMath::Prob(chi2,ndf)` in ROOT)

$$p(\mathrm{LS}(\hat{\theta}), \mathrm{ndf}) = \int_{\mathrm{LS}(\hat{\theta})}^{\infty} \chi^2(t|\mathrm{ndf})dt$$

*e.g.*, $p(3,2) \approx 22\%$
$p(300,200) \approx 6\times10^{-6}$

- Does the $\chi^2$ capture the full goodness-of-fit? Not really, being insensitive to the sign of the deviation between the data and the model. The $\chi^2$ test can be complemented with the runs test of the deviations

# Small uncertainties do not imply a good fit (nor viceversa)

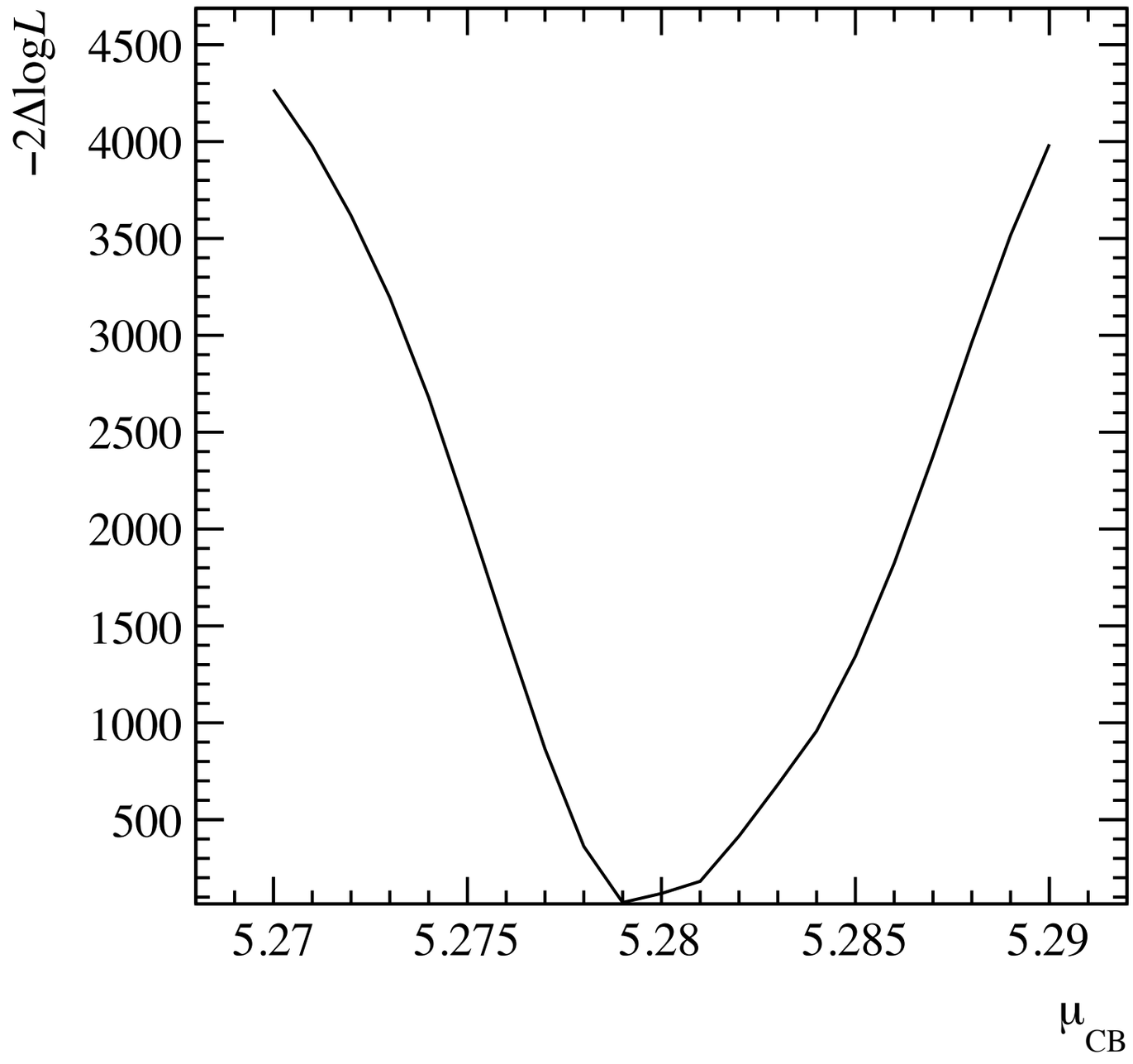
Small statistical uncertainties do not mean the fit is good, nor viceversa:

- Uncertainty size is driven by the curvature of LS/–log$L$ near its minimum
- For LS goodness of fit is driven by the actual value at minimum $LS_{min} \sim \chi^2$

Variance of estimator (*i.e.*, statistical uncertainty on the estimate) tells us about the spread in values of the estimator if one repeats the estimate many times on independent samples

Goodness of fit ($\chi^2$ $p$-value) tells us what fraction of repeated experiments will give equal of worse agreement with model according to $LS_{min}$ and assuming that the hypothesis is correct. Low $p$-value suggest incorrect model ( ==> systematic uncertainty)

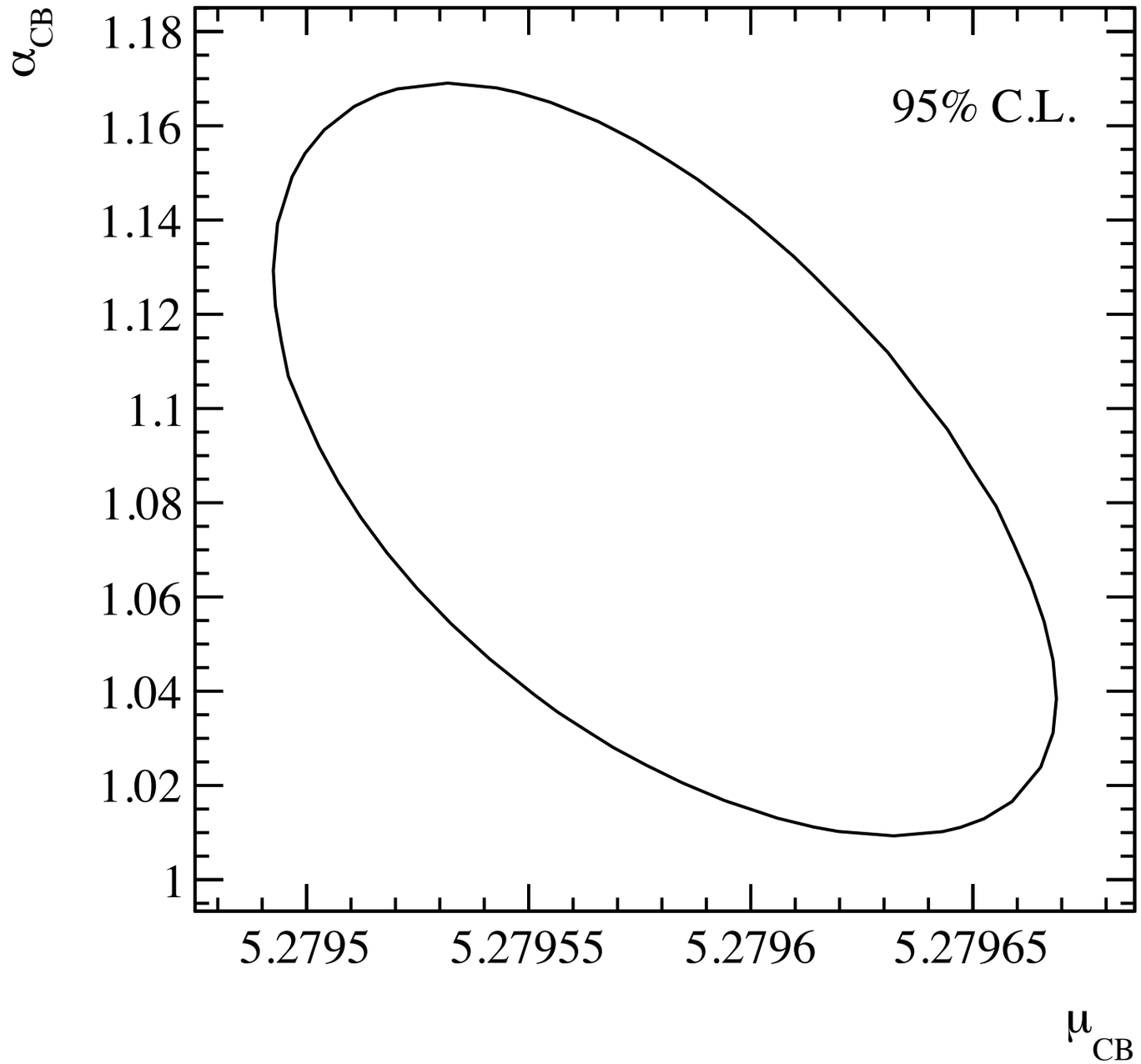
# Likelihood profile in 1D



# Likelihood profile in 1D

```
223         // Likelihood profile
224         std::cout << "Plotting likelihood profile vs " << fitter->GetParName(1) << "..." << std::endl;
225         double minFCN(fmin);
226         double fcn[21], par[21];
227         double pmin(5.27), pstep(0.001);
228         unsigned int n(0);
229         char name[20];
230         double oldvalue, err, low, high;
231         for (unsigned int i=0; i<21; ++i) {
232            fitter->GetParameter(1,name,oldvalue,err,low,high);
233            fitter->SetParameter(1,name,pmin+i*pstep,0.,low,high);
234            if ( 0 != fitter->ExecuteCommand("MIGRAD",arglist,2) ) continue;
235            minuit->mnstat(fmin,fedm,errdef,npari,nparx,istat);
236            par[n] = fitter->GetParameter(1);
237            fcn[n] = fmin;
238            if (fcn[n] < minFCN) {
239               minFCN = fcn[n];
240               std::cout << "Found new global minimum." << std::endl;
241            }
242            n++;
243         }
244         for (unsigned int i=0; i<n; ++i) fcn[i] -= minFCN;
245
246         auto g_profile = new TGraph(n,par,fcn);
247         g_profile->SetTitle("");
248         g_profile->GetXaxis()->SetTitle(fitter->GetParName(1));
249         g_profile->GetYaxis()->SetTitle("#minus2#Deltalog#it{L}");
250         g_profile->SetLineWidth(2);
```

# Likelihood profile in 2D (CONTOUR)



# Likelihood profile in 2D (CONTOUR)

```
196         // 2D contour
197         std::cout << "Plotting 95% CL contour in (" << fitter->GetParName(1) << "," <<
               fitter->GetParName(3) << ") plane..." << std::endl;
198         printlevel = -1;
199         fitter->ExecuteCommand("SET PRI",&printlevel,1);
200         fitter->ExecuteCommand("SET NOW",&printlevel,0);
201
202         fitter->SetErrorDef( ROOT::Math::chisquared_quantile(0.95,2) );
203         auto g_cont = (TGraph*) minuit->Contour(60,1,3);
204         fitter->SetErrorDef(errdef);
205
206         TCanvas c2("c2","c2");
207         if (g_cont) {
208            g_cont->SetTitle("");
209            g_cont->GetXaxis()->SetTitle(fitter->GetParName(1));
210            g_cont->GetYaxis()->SetTitle(fitter->GetParName(3));
211            g_cont->SetLineWidth(2);
212            g_cont->Draw("al");
213
214            Plotter::plot_text("95% C.L.",0.9,0.88);
215
216            outfilename = args.outdir + "simple-1d-fit_contour.pdf";
217            c2.Print(outfilename.c_str());
218         } else {
219            std::cout << "Contour failed" << std::endl;
220            c2.Close();
221         }
```

# Likelihood profile in 2D (CONTOUR)

| Number of Parameters | Confidence level (probability contents desired inside hypercontour of $\chi^2 = \chi^2_{\text{min}} + \texttt{UP}$) | | | | |
|---|---|---|---|---|---|
| | 50% | 70% | 90% | 95% | 99% |
| 1 | 0.46 | 1.07 | 2.70 | 3.84 | 6.63 |
| 2 | 1.39 | 2.41 | 4.61 | 5.99 | 9.21 |
| 3 | 2.37 | 3.67 | 6.25 | 7.82 | 11.36 |
| 4 | 3.36 | 4.88 | 7.78 | 9.49 | 13.28 |
| 5 | 4.35 | 6.06 | 9.24 | 11.07 | 15.09 |
| 6 | 5.35 | 7.23 | 10.65 | 12.59 | 16.81 |
| 7 | 6.35 | 8.38 | 12.02 | 14.07 | 18.49 |
| 8 | 7.34 | 9.52 | 13.36 | 15.51 | 20.09 |
| 9 | 8.34 | 10.66 | 14.68 | 16.92 | 21.67 |
| 10 | 9.34 | 11.78 | 15.99 | 18.31 | 23.21 |
| 11 | 10.34 | 12.88 | 17.29 | 19.68 | 24.71 |
| | If FCN is $-\log(\text{likelihood})$ instead of $\chi^2$, all values of UP should be divided by 2. | | | | |

# Example 2: efficiency fit

# Frequentist approach

- Let's consider a simple counting experiment where $N_{\text{pass}}$ is the number of events satisfying a given selection criteria out of a total of $N$ events

- The estimated efficiency is

$$\hat{\varepsilon} = \frac{N_{\text{pass}}}{N}$$

- Since the process of applying some selection criteria is binomial (the event can either pass or fail the criteria) with probability equal to the "true" efficiency $\varepsilon$, the estimated statistical uncertainty is

$$\hat{\sigma}_{\varepsilon} = \sqrt{\frac{\hat{\varepsilon}(1-\hat{\varepsilon})}{N}}$$

- This is the most used approach to compute the uncertainty on the efficiency, when sufficiently far away from the limiting cases in which $\hat{\varepsilon}$=0 or 1 (for other approaches see, e.g., ROOT's `TEfficiency`)

# Alternative derivation

- Changing point of view, let's consider the number of events passing ($N_{\text{pass}}$) or failing ($N_{\text{fail}} = N - N_{\text{pass}}$) the selection criteria as independent Poisson countings

$$\hat{\varepsilon} = \frac{N_{\text{pass}}}{N_{\text{pass}} + N_{\text{fail}}}, \quad \hat{\sigma}_{N_{\text{pass/fail}}} = \sqrt{N_{\text{pass/fail}}}$$

$$\hat{\sigma}_{\varepsilon} = \frac{\sqrt{N_{\text{pass}}^2 \sigma_{N_{\text{fail}}}^2 + N_{\text{fail}}^2 \sigma_{N_{\text{pass}}}^2}}{(N_{\text{pass}} + N_{\text{fail}})^2} = \frac{\sqrt{N_{\text{pass}}^2 N_{\text{fail}} + N_{\text{fail}}^2 N_{\text{pass}}}}{(N_{\text{pass}} + N_{\text{fail}})^2}$$

$$= \sqrt{\frac{\hat{\varepsilon}(1-\hat{\varepsilon})}{N}}$$

# Alternative derivation: why does it work?

$$\begin{array}{l} x \sim \text{Poisson}(\mu_x) \\ y \sim \text{Poisson}(\mu_y) \end{array} \quad \Longrightarrow \quad P(x = k | x+y = n) = \text{Binomial}\left(k; n, \varepsilon = \frac{\mu_x}{\mu_x + \mu_y}\right)$$

Proof:

$$\begin{aligned} P(x = k|x + y = n) &= \frac{P(x = k)P(x + y = n|x = k)}{P(x + y = n)} \\ &= \frac{\frac{\mu_x^k}{k!}\, e^{-\mu_x}\, \frac{\mu_y^{n-k}}{(n-k)!}\, e^{-\mu_y}}{\frac{(\mu_x+\mu_y)^n}{n!}\, e^{-(\mu_x+\mu_y)}} = \frac{n!}{k!(n-k)!}\, \frac{\mu_x^k \mu_y^{n-k}}{(\mu_x+\mu_y)^n} \\ &= \frac{n!}{k!(n-k)!}\, \frac{\mu_x^k \mu_y^{n-k}}{(\mu_x+\mu_y)^{n-k+k}} = \frac{n!}{k!(n-k)!} \left(\frac{\mu_x}{\mu_x+\mu_y}\right)^k \left(\frac{\mu_y}{\mu_x+\mu_y}\right)^{n-k} \\ &= \binom{n}{k} \varepsilon^k (1-\varepsilon)^{n-k} \end{aligned}$$

# What if it’s not just counting?

- All the above holds for a simple counting experiments. Often, however, we need to determine the number of “signal” events out of a sample where also background is present

- Typically this is done by separating signal and background on a statistical basis using a fit to some distribution that gives enough discrimination power

- The fit returns an uncertainty on the number of signal events that includes also the uncertainty due to the background subtraction

# Uncertainty on the efficiency in the case of fitting

- Among possible correct methods, the easiest is to use the alternative approach: *i.e.*, fit the independent samples that pass and fail the selection criteria and compute

$$\hat{\sigma}_\varepsilon = \frac{\sqrt{\hat{N}_{\mathrm{pass}}^2 \hat{\sigma}_{N_{\mathrm{fail}}}^2 + \hat{N}_{\mathrm{fail}}^2 \hat{\sigma}_{N_{\mathrm{pass}}}^2}}{(\hat{N}_{\mathrm{pass}} + \hat{N}_{\mathrm{fail}})^2}$$

- Other approaches have been suggested/used, some of which may occasionally even work, but they generally misestimate the uncertainty (*e.g.*, see this talk)

- Even better would be to perform a simultaneous fit with $\varepsilon$ as shared parameter, *e.g.*,

```
./bin/efficiency-fit -p -g -n 1000000 -o output/
```

# Efficiency fit

- Simultaneous least-squares fit to the mass distributions of $D^0 \to K^-\pi^+\pi^-\pi^+$ decay candidates passing and failing the requirement $p(K^-) > 1$ GeV/$c$, to determine the efficiency of the requirement on both signal and background decays.

- The data consists pseudo data generated from some assumed mass model and input efficiencies

- Two histograms are filled: the first with candidates passing the kaon-momentum requirement ($h_{\text{pass}}$), the second with candidates failing the requirement ($h_{\text{fail}}$)

# The function to minimize: FCN

$$\mathrm{LS}(\vec{\theta}_{\mathrm{pass}}, \vec{\theta}_{\mathrm{fail}}) = \sum_{i \in h_{\mathrm{pass}}} \left( \frac{n_i - n_i^{\mathrm{pass}}(\vec{\theta}_{\mathrm{pass}})}{\sigma_i} \right)^2 + \sum_{i \in h_{\mathrm{fail}}} \left( \frac{n_i - n_i^{\mathrm{fail}}(\vec{\theta}_{\mathrm{fail}})}{\sigma_i} \right)^2$$

```
71  // Computation of least squares
72  double chi2(0.); int ndf(0);
73
74  void leastSquares(int &/*npar*/, double * /*gin*/, double &result, double *pars, int /* flag */)
75  {
76     chi2 = 0.; ndf = 0;
77
78     // compute least squares
79     for (unsigned int i=1; i<nbins; ++i) {
80        double m = hpass->GetBinCenter(i);
81
82        double epass = hpass->GetBinError(i);
83        if (epass!=0.) {
84           double res = hpass->GetBinContent(i) - fpass(&m,pars);
85           chi2 += res*res/epass/epass;
86           ndf++;
87        }
88
89        double efail = hfail->GetBinError(i);
90        if (efail!=0.) {
91           double res = hfail->GetBinContent(i) - ffail(&m,pars);
92           chi2 += res*res/efail/efail;
93           ndf++;
94        }
95     }
96
97     result = chi2;
98  }
```

# Predicted number of candidates (*i.e.*, the fit function)

- Mass distribution of signal and background candidates described by

$$\mathrm{pdf}_{\mathrm{sgn}}(m|\mu,\sigma) \propto e^{-\frac{1}{2}\left(\frac{m-\mu}{\sigma^2}\right)^2}$$

$$\mathrm{pdf}_{\mathrm{bkg}}(m|\lambda) \propto e^{-\lambda m}$$

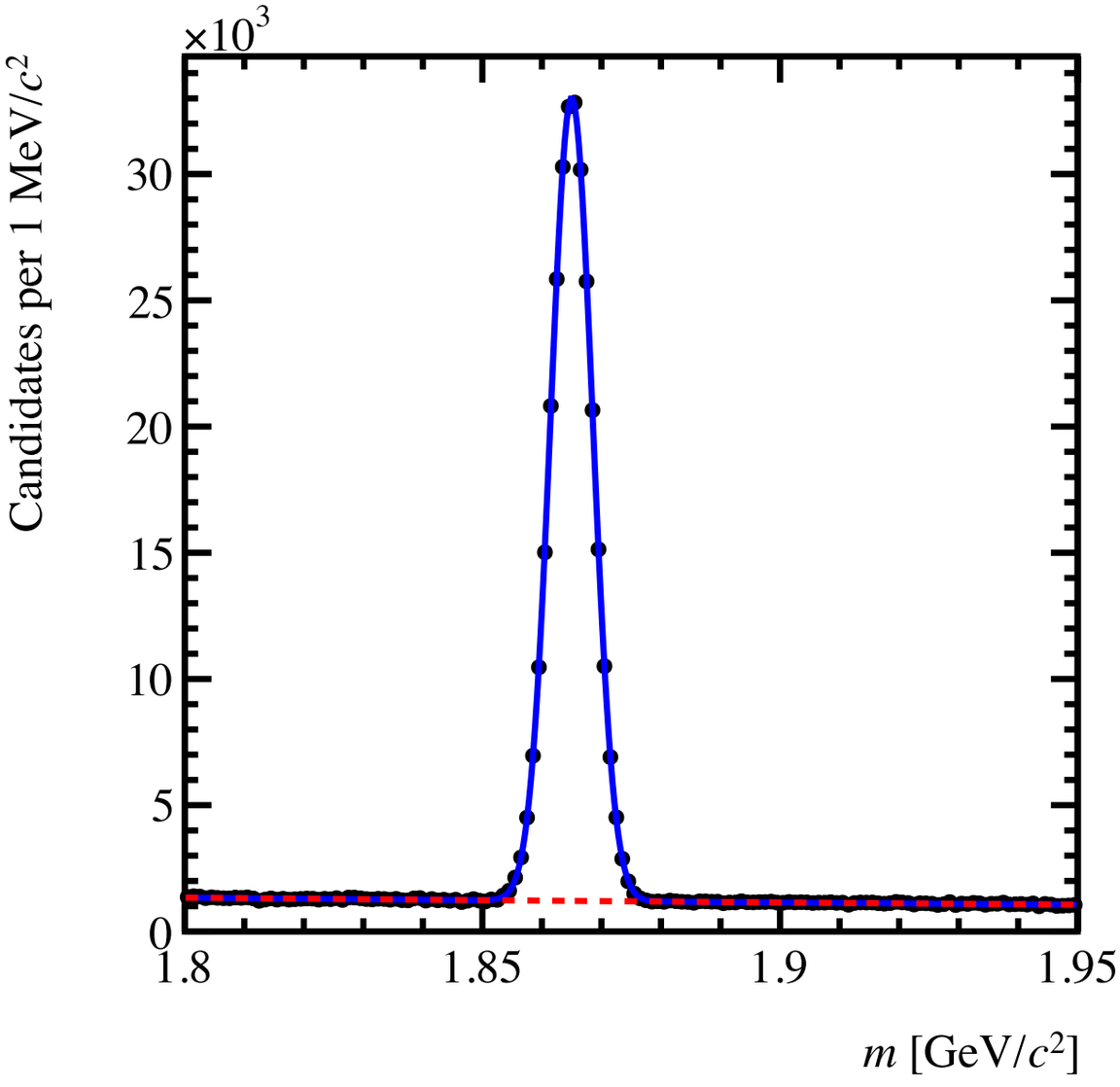
- Shapes for candidates passing/failing the selection could be different: *e.g.*, assume different peak resolution for signal and different slopes for background

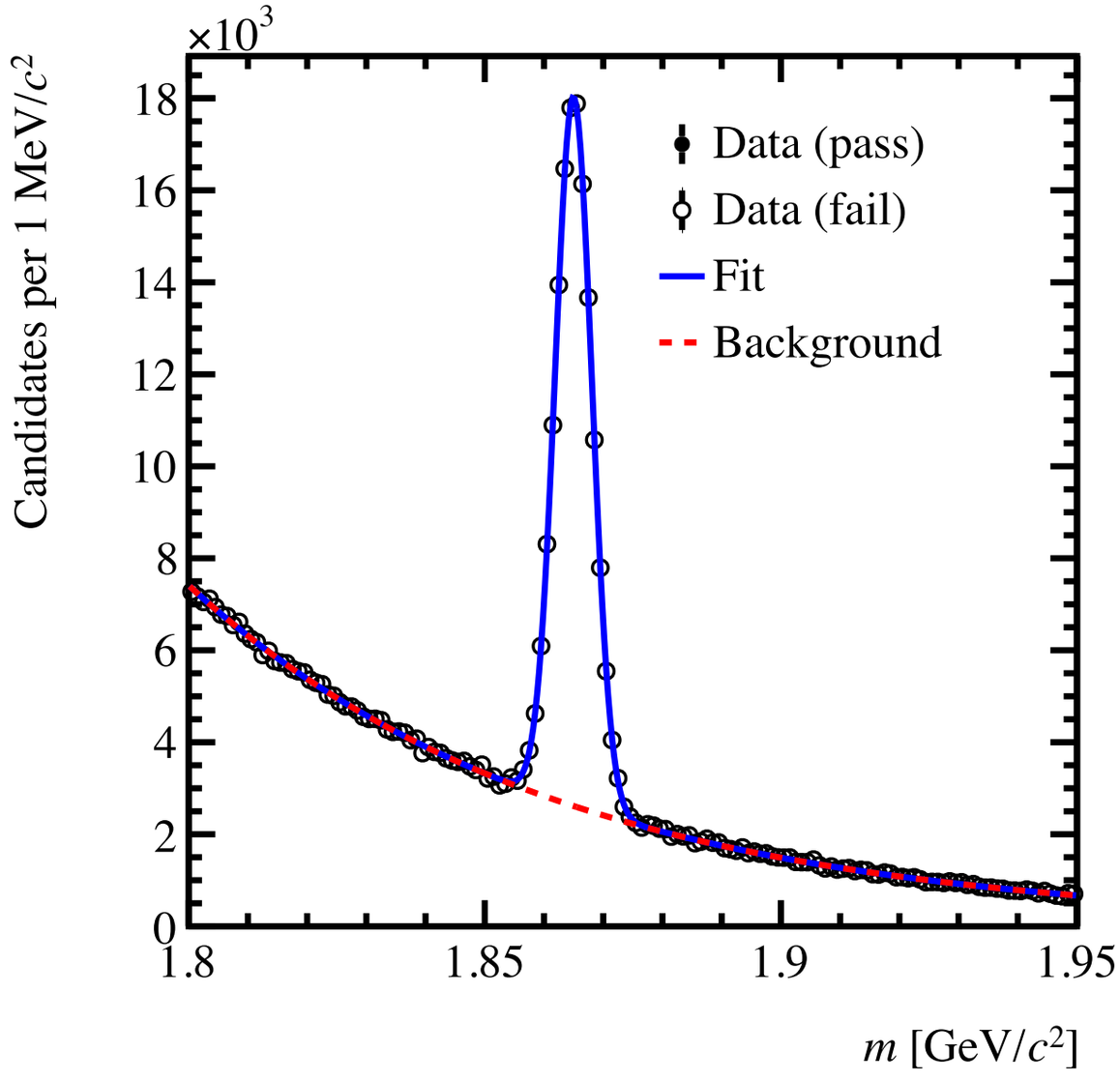
$$n_i^{\mathrm{pass}}(N_{\mathrm{sgn}},\epsilon_{\mathrm{sgn}},\mu,\sigma_{\mathrm{pass}},N_{\mathrm{bkg}},\epsilon_{\mathrm{bkg}},\lambda_{\mathrm{pass}}) = N_{\mathrm{sgn}}\epsilon_{\mathrm{sgn}}\mathrm{pdf}_{\mathrm{sgn}}(m_i|\mu,\sigma_{\mathrm{pass}}) + N_{\mathrm{bkg}}\epsilon_{\mathrm{bkg}}\mathrm{pdf}_{\mathrm{bkg}}(m_i|\lambda_{\mathrm{pass}})$$

$$n_i^{\mathrm{fail}}(N_{\mathrm{sgn}},\epsilon_{\mathrm{sgn}},\mu,\sigma_{\mathrm{fail}},N_{\mathrm{bkg}},\epsilon_{\mathrm{bkg}},\lambda_{\mathrm{fail}}) = N_{\mathrm{sgn}}(1-\epsilon_{\mathrm{sgn}})\mathrm{pdf}_{\mathrm{sgn}}(m_i|\mu,\sigma_{\mathrm{fail}}) + N_{\mathrm{bkg}}(1-\epsilon_{\mathrm{bkg}})\mathrm{pdf}_{\mathrm{bkg}}(m_i|\lambda_{\mathrm{fail}})$$

```
**********
**    6 **HESSE        5000         1
**********
COVARIANCE MATRIX CALCULATED SUCCESSFULLY
FCN=264.632 FROM HESSE     STATUS=OK             75 CALLS        1488 TOTAL
                    EDM=2.10224e-06    STRATEGY= 2      ERROR MATRIX ACCURATE
 EXT PARAMETER                                   INTERNAL      INTERNAL
 NO.   NAME      VALUE            ERROR          STEP SIZE       VALUE
  1  N_{sig}          3.99902e+05   6.95535e+02   8.60820e-08  -1.44422e+00
  2  #epsilon_{sig}   6.99934e-01   8.40303e-04   1.39374e-06   4.11372e-01
  3  N_{bkg}          5.99500e+05   8.28125e+02   8.43860e-08  -1.41576e+00
  4  #epsilon_{bkg}   2.98510e-01   6.37407e-04   1.08683e-06  -4.14770e-01
  5  #mu              1.86499e+00   6.25911e-06   6.69834e-08  -1.33826e-01
  6  #sigma_{pass}    3.51101e-03   5.99840e-06   8.65590e-08  -1.45222e+00
  7  #sigma_{fail}    3.10719e-03   1.05031e-05   2.86439e-08  -1.45925e+00
  8  #lambda_{pass}   1.53915e+00   5.53262e-02   4.39965e-05   1.53915e+00
  9  #lambda_{fail}   1.60332e+01   4.14499e-02   6.55460e-06   1.60332e+01
EXTERNAL ERROR MATRIX.    NDIM=  25    NPAR=  9    ERR DEF=1
 4.838e+05 -7.077e-02 -8.419e+04 -9.017e-03 -3.501e-05  4.121e-04  1.193e-03 -6.242e-01  1.564e+00
-7.077e-02  7.061e-07  7.101e-02 -7.356e-08  6.602e-11  3.088e-10 -2.087e-09 -4.532e-07 -2.737e-06
-8.419e+04  7.101e-02  6.858e+05  1.000e-02  3.438e-05 -4.139e-04 -1.197e-03  3.750e-01 -1.684e+00
-9.017e-03 -7.356e-08  1.000e-02  4.063e-07 -1.122e-11 -4.841e-10  5.958e-10  4.303e-07  8.388e-07
-3.501e-05  6.602e-11  3.438e-05 -1.122e-11  3.918e-11 -1.697e-13 -1.092e-12  6.427e-09  2.427e-09
 4.121e-04  3.088e-10 -4.139e-04 -4.841e-10 -1.697e-13  3.598e-11  4.731e-15 -8.368e-09 -1.052e-11
 1.193e-03 -2.087e-09 -1.197e-03  5.958e-10 -1.092e-12  4.731e-15  1.103e-10 -1.791e-10  3.472e-08
-6.242e-01 -4.532e-07  3.750e-01  4.303e-07  6.427e-09 -8.368e-09 -1.791e-10  3.061e-03  3.982e-07
 1.564e+00 -2.737e-06 -1.684e+00  8.388e-07  2.427e-09 -1.052e-11  3.472e-08  3.982e-07  1.718e-03
 PARAMETER  CORRELATION COEFFICIENTS
       NO.  GLOBAL      1      2      3      4      5      6      7      8      9
        1  0.24298   1.000 -0.121 -0.146 -0.020 -0.008  0.099  0.163 -0.016  0.054
        2  0.29607  -0.121  1.000  0.102 -0.137  0.013  0.061 -0.237 -0.010 -0.079
        3  0.21382  -0.146  0.102  1.000  0.019  0.007 -0.083 -0.138  0.008 -0.049
        4  0.19679  -0.020 -0.137  0.019  1.000 -0.003 -0.127  0.089  0.012  0.032
        5  0.02970  -0.008  0.013  0.007 -0.003  1.000 -0.005 -0.017  0.019  0.009
        6  0.18519   0.099  0.061 -0.083 -0.127 -0.005  1.000  0.000 -0.025 -0.000
        7  0.30096   0.163 -0.237 -0.138  0.089 -0.017  0.000  1.000 -0.000  0.080
        8  0.03678  -0.016 -0.010  0.008  0.012  0.019 -0.025 -0.000  1.000  0.000
        9  0.11364   0.054 -0.079 -0.049  0.032  0.009 -0.000  0.080  0.000  1.000
chi2/ndf = 264.632/289
Probability = 0.845097
```

# Fit projections



# Validate fit with pseudoexperiments

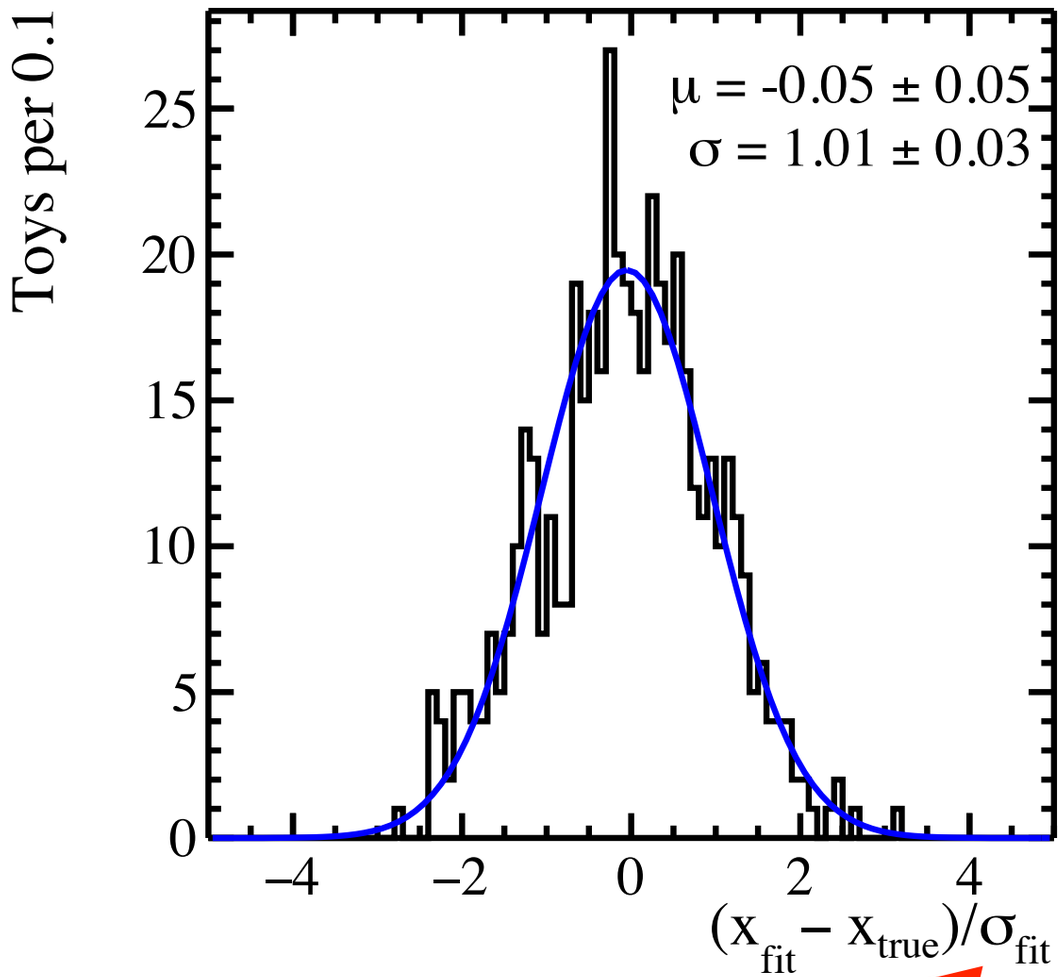
- Use pseudoexperiments (*a.k.a.* toy Monte Carlo simulation) drawn from the PDF to understand the distribution of the fit estimators and their properties prior to applying them to data

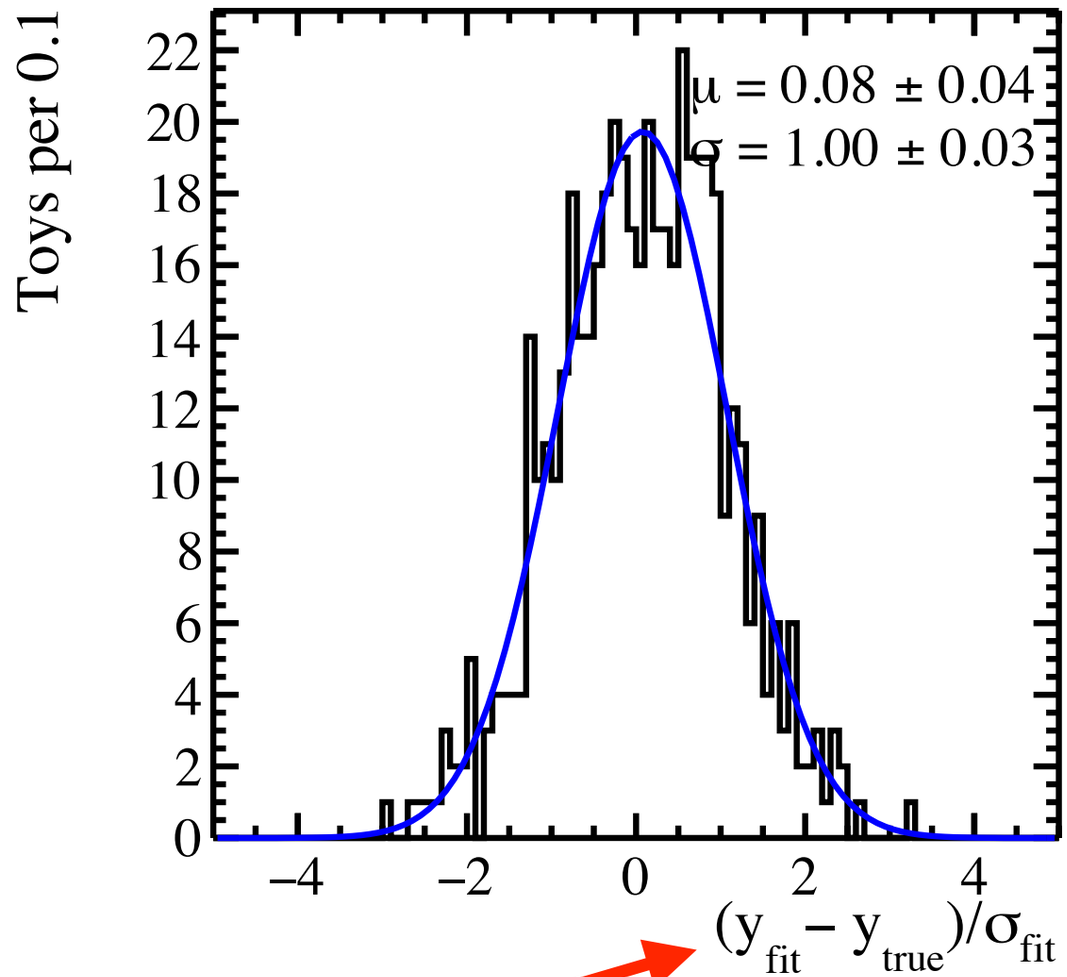
- Choose a plausible true value of the relevant parameters *m* and generate several sets of simulated data *x* from random numbers distributed according to $p(x|m)$

- Run the fit on each set (that is, repeat the experiment) and look at the distribution of the estimator

- Repeat for all relevant choices of true values for *m* (important and often overlooked)

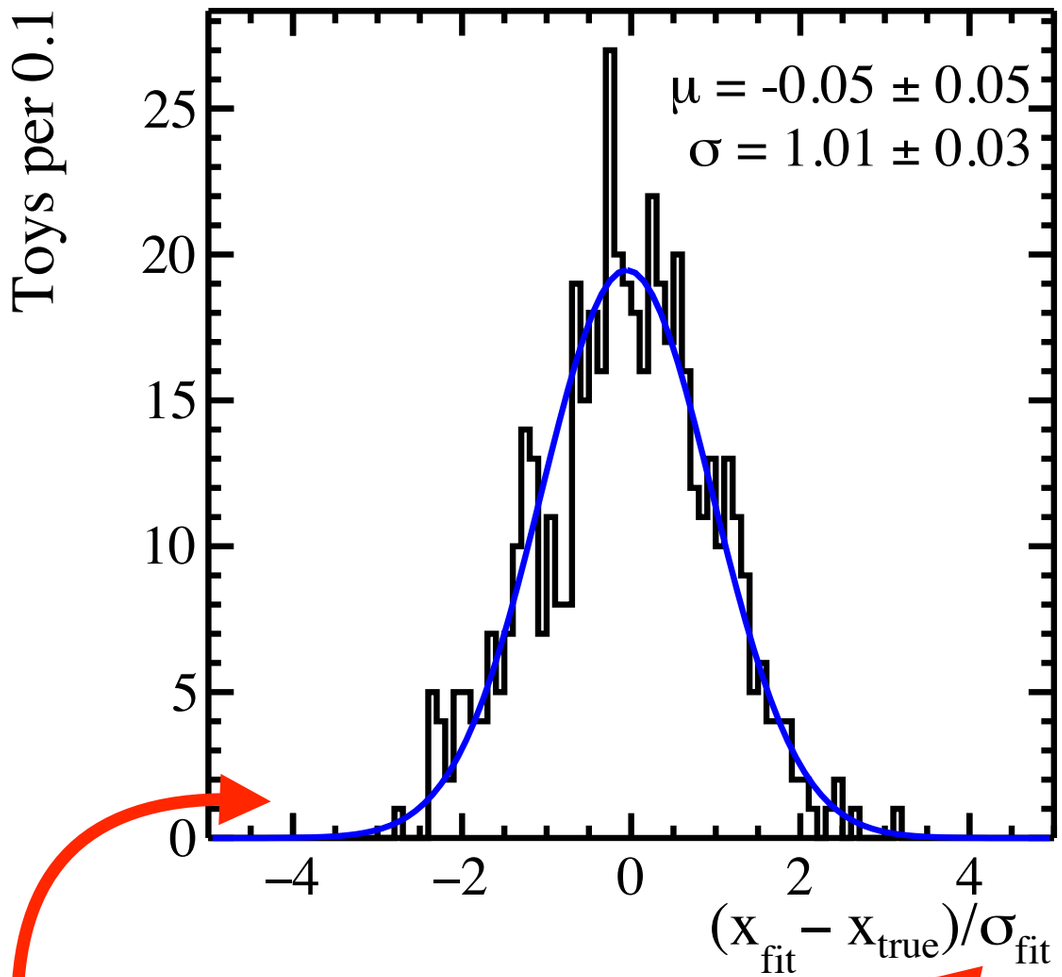
# A standard diagnostics — fit pulls

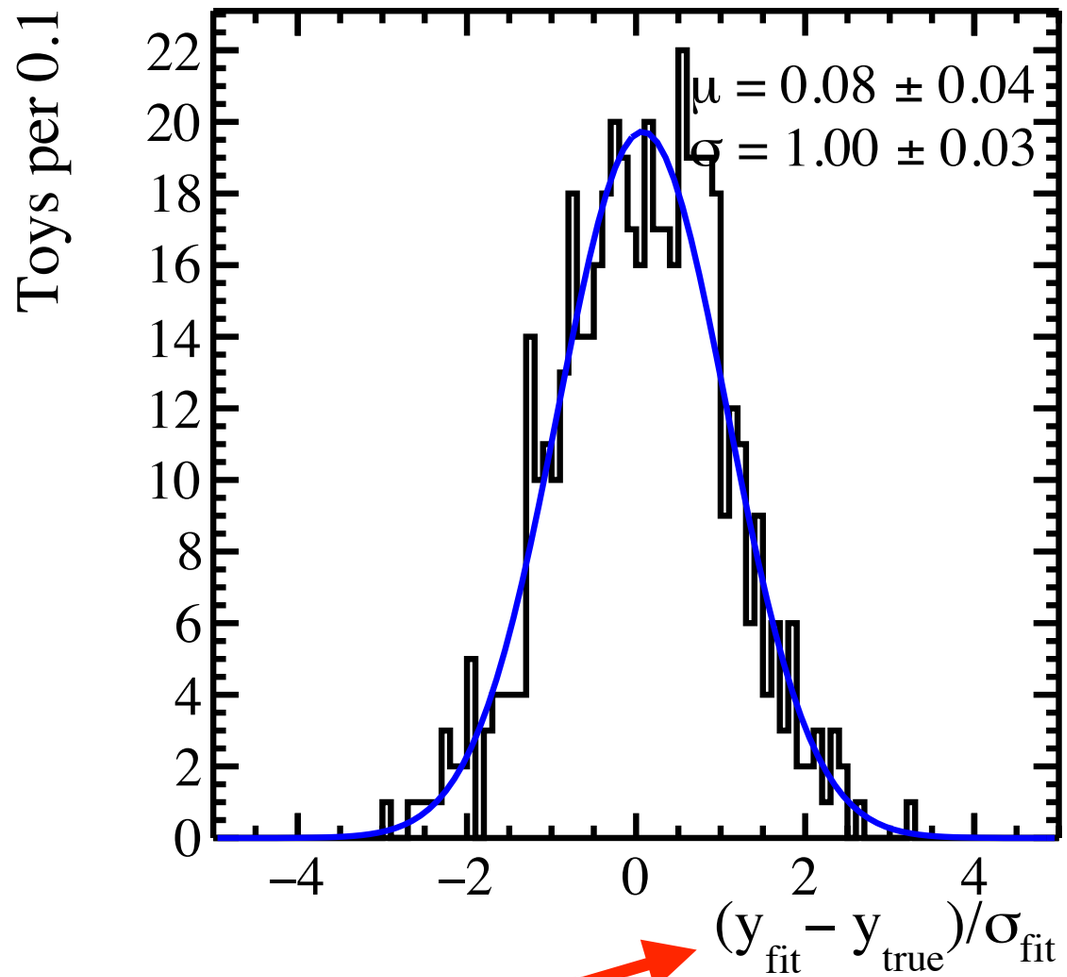



Distribution of the difference between fit estimate and the true value of the parameter, divided by the estimate of the std dev.

# A standard diagnostics — fit pulls

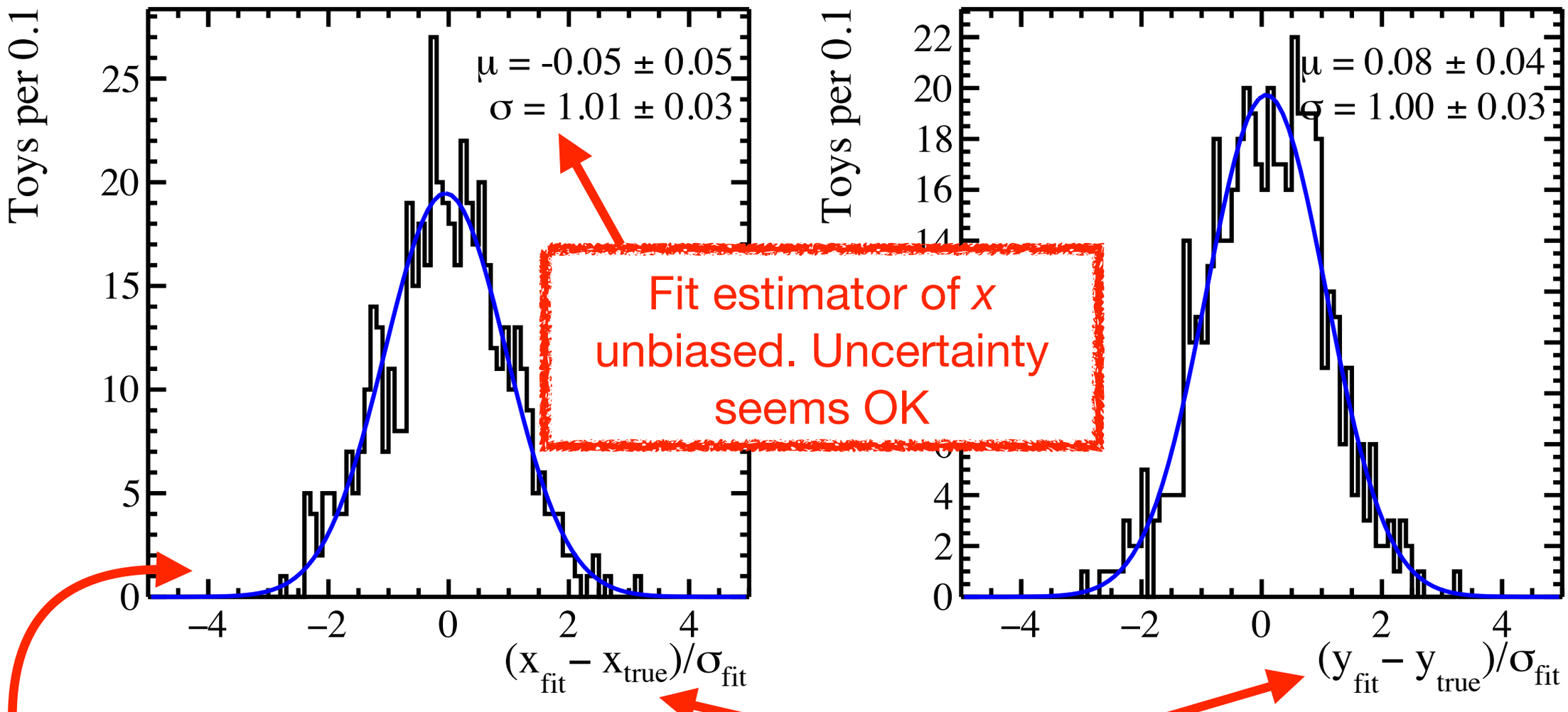



Each entry is a simulated experiment, generated with the same set of true parameters.

Distribution of the difference between fit estimate and the true value of the parameter, divided by the estimate of the std dev.

# A standard diagnostics — fit pulls

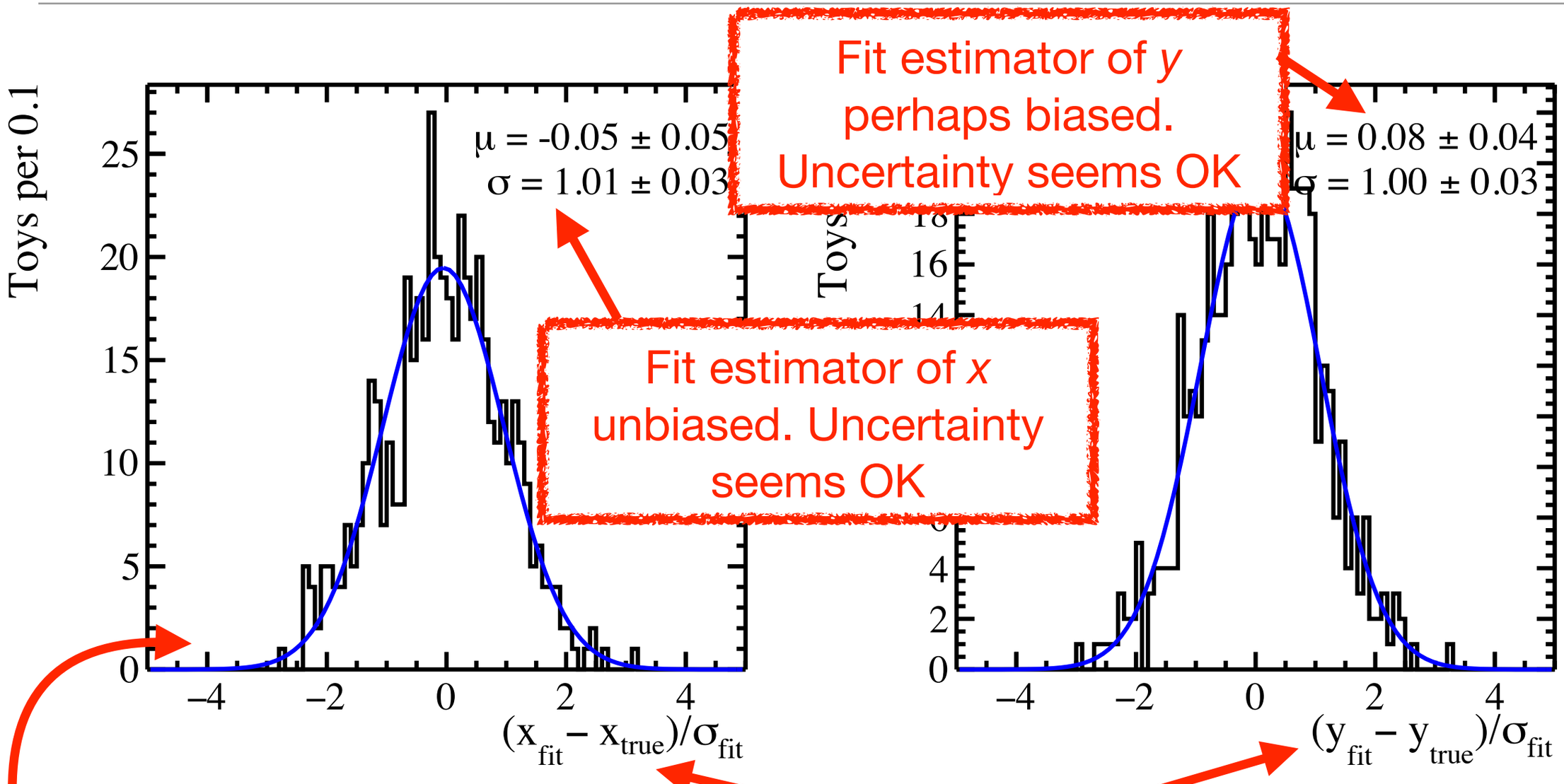



Fit estimator of *x* unbiased. Uncertainty seems OK

Each entry is a simulated experiment, generated with the same set of true parameters.

Distribution of the difference between fit estimate and the true value of the parameter, divided by the estimate of the std dev.

# A standard diagnostics — fit pulls




Fit estimator of *y* perhaps biased. Uncertainty seems OK

Fit estimator of *x* unbiased. Uncertainty seems OK

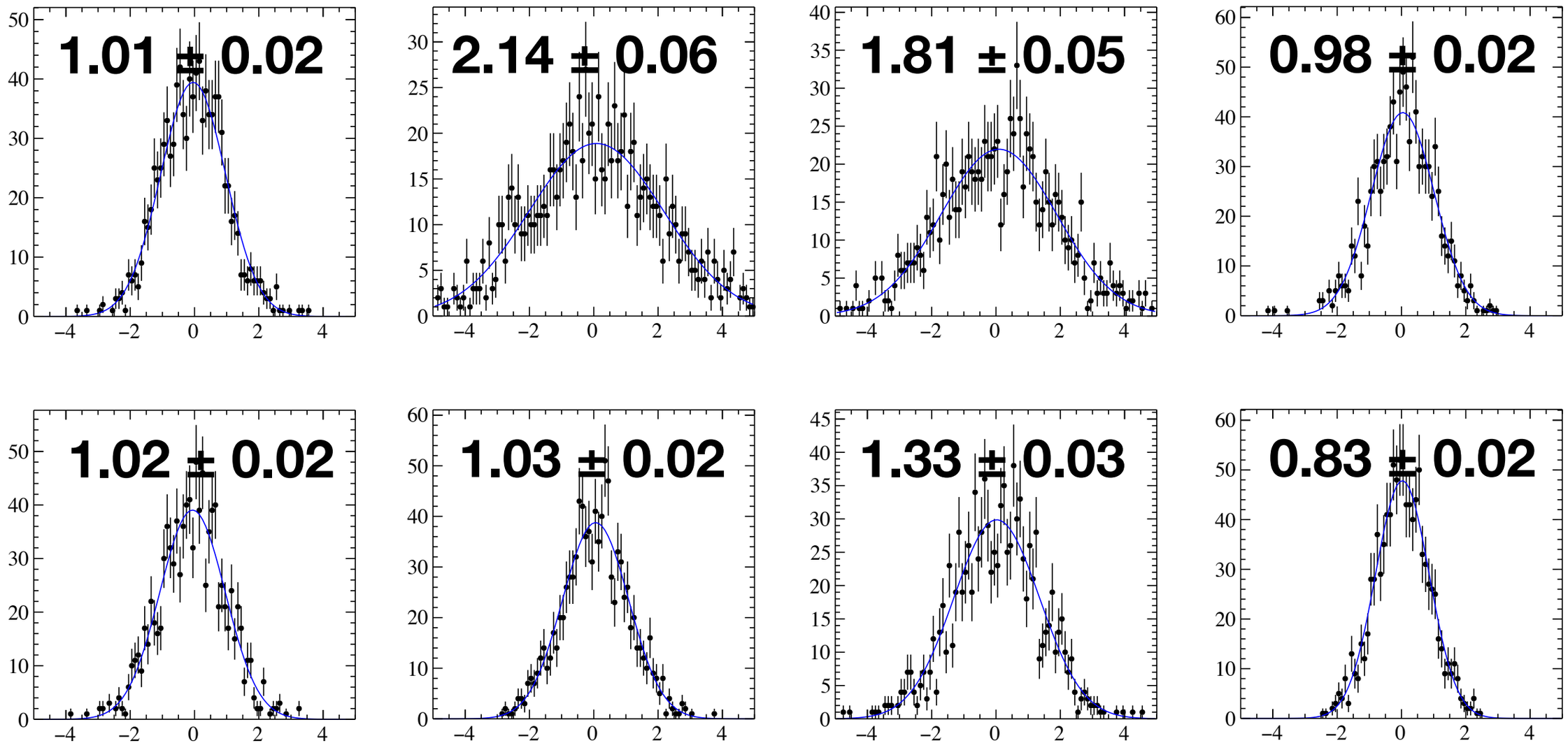
Each entry is a simulated experiment, generated with the same set of true parameters.

Distribution of the difference between fit estimate and the true value of the parameter, divided by the estimate of the std dev.

# Validate for any possible true fit parameters

Different methods to compute the uncertainty on the efficiency

Different true values for fit parameters




Fit pulls $\frac{\hat{\varepsilon} - \varepsilon}{\hat{\sigma}_{\varepsilon}}$ (**pull width**)

# Pseudoexperiments

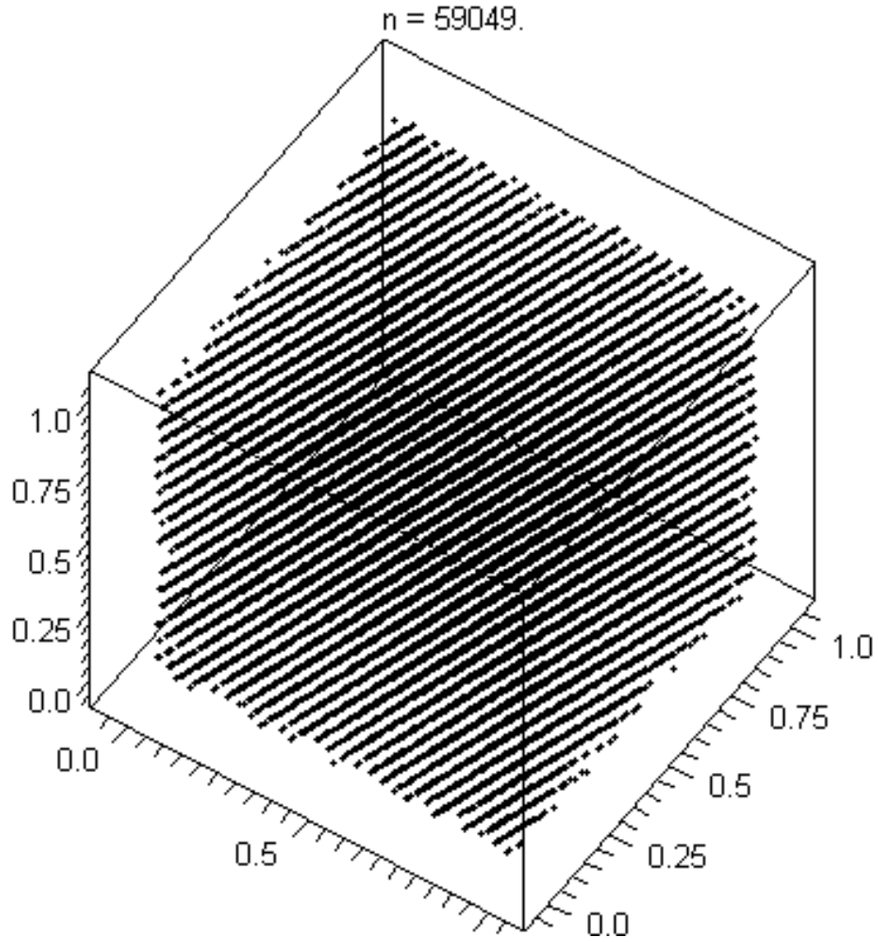
- Use numerical methods for sampling distributions of relevant quantities (*e.g.*, observables, estimators, likelihood ratios, *etc*) to then calculate probabilities and related quantities
- Typically it boils down to:
  - Generate a random sequence $r_1$, $r_2$, …, $r_n$ uniformly distributed in [0,1]
  - Use this to produce another sequence $x_1$, $x_2$, …, $x_n$ distributed according to some PDF, $p(x)$ — $x$ can be multidimensional
  - Use the $x$ values to estimate the properties of $p(x)$, *e.g.*, mean, variance, fraction of values with $a < x < b$, *etc*
- Genuinely random numbers can be extracted from physics processes (time intervals between radioactive decays, thermal noise across a resistor, atmospheric radio noise). Usually tabulated in large databases and impractical to use (*e.g.*, www.random.org)

# Pseudorandom number generators

- Typically we use sequences of pseudorandom numbers. If properly handled, they are indistinguishable from true random numbers and are reproducible, which is useful for debugging

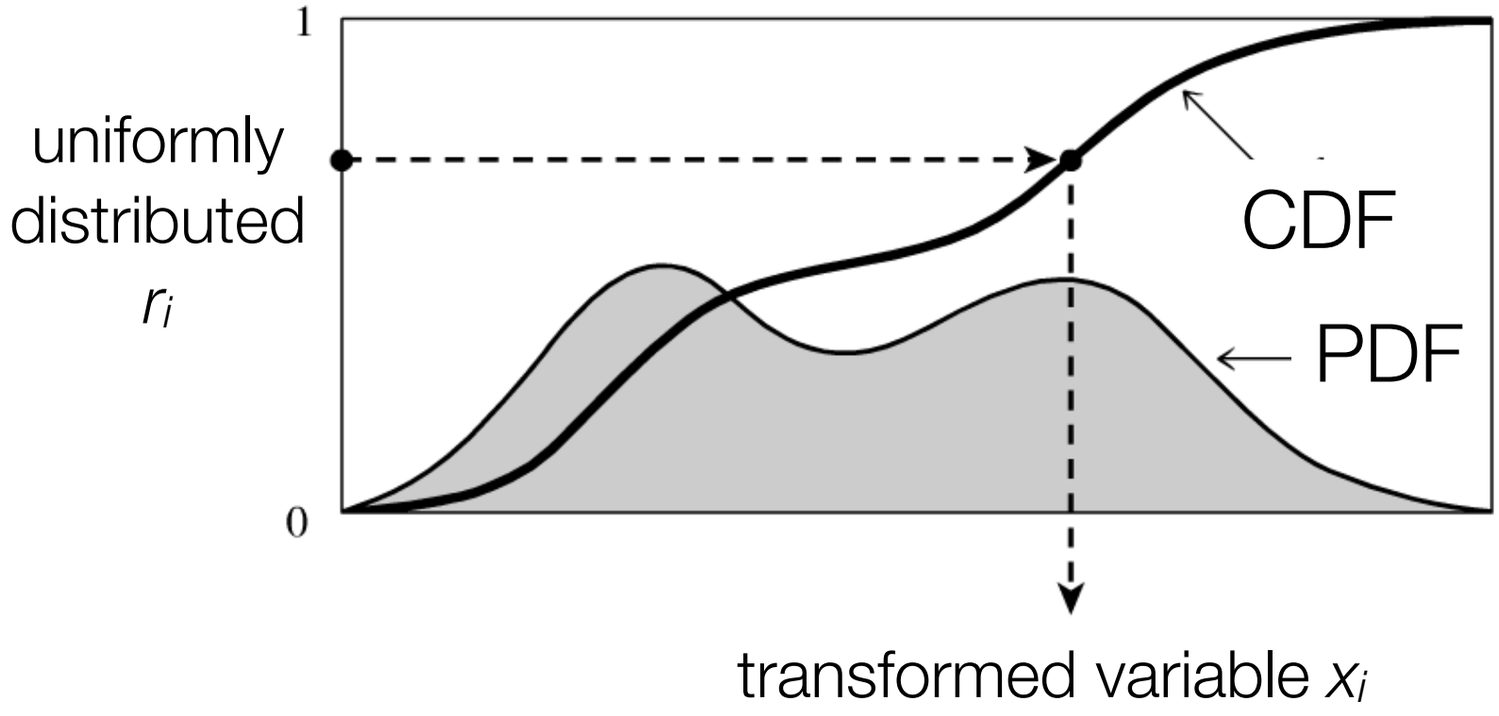
- Various available with different performances (*e.g.*, mid-square method, linear congruential generators, Mersenne Twister algorithm, *etc*). As usual, understand what you're doing

- In practice, `TRandom3` from ROOT (based on the Mersenne Twister algorithm) is relatively fast and sufficiently random for most HEP usages




Triplets generated with LCG

# From uniform to *p*(x): the inverse cumulative method



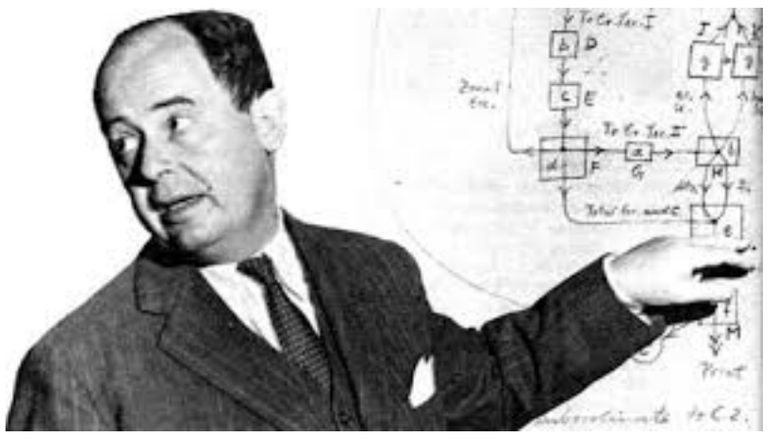

- Generate $r_i$ uniformly distributed in [0,1]
- Compute $x_i$ such that $r_i = \mathrm{CDF}(x_i)$ where $\mathrm{CDF}(x) = \int_0^x p(t)dt$
  - That is $x_i = \mathrm{CDF}^{-1}(r_i)$

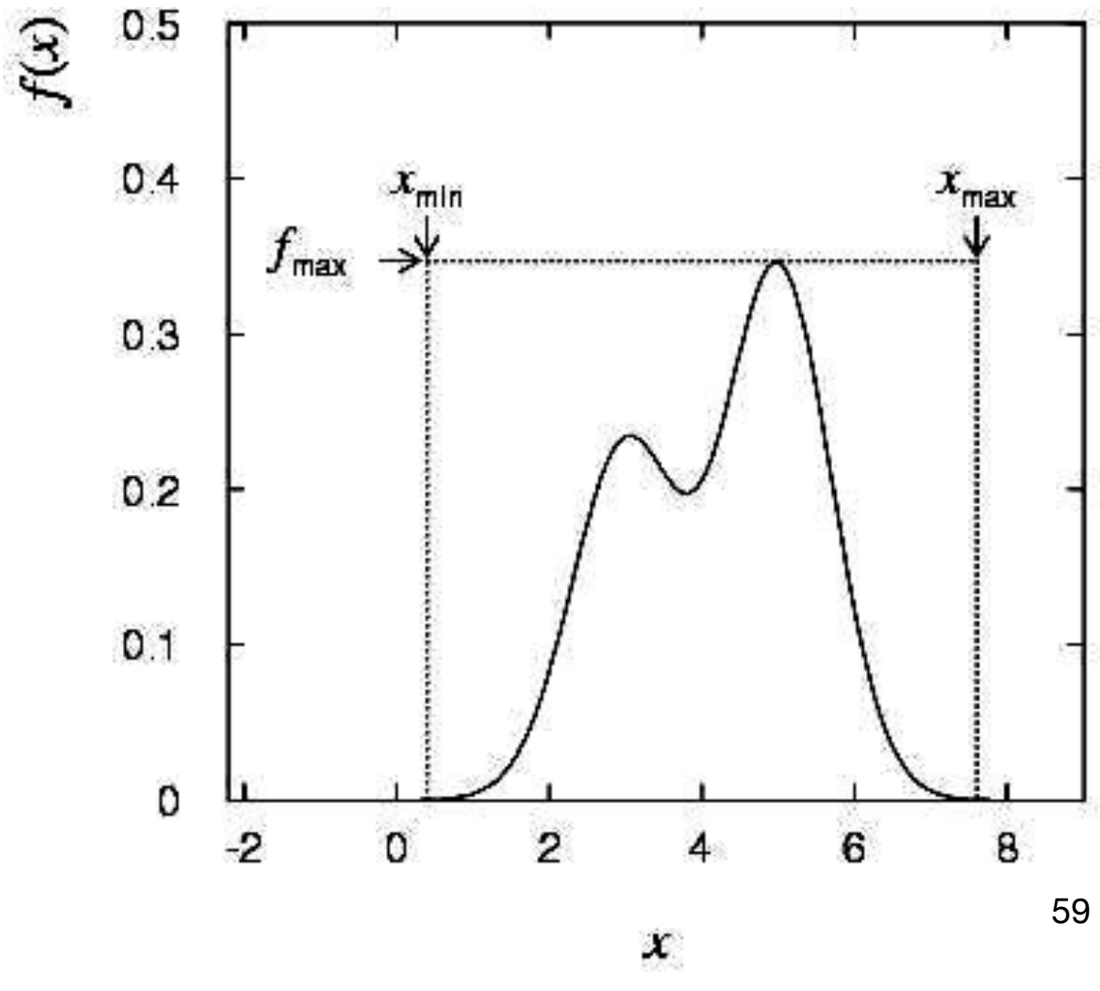
# From uniform to $p$(x): the (von Neumann) accept-reject method

- Enclose the $f(x)$ in a "box"
  - Generate $x_i$ uniformly in $[x_{min}, x_{max}]$ starting from $r_i$ uniform in [0,1]

$$x_i = x_{min} + r_i\,(x_{max} - x_{min})$$

- Generate $y_i$ uniformly in $[0, f_{max}]$
- If $y_i < f(x)$, then accept $x_i$, otherwise reject it



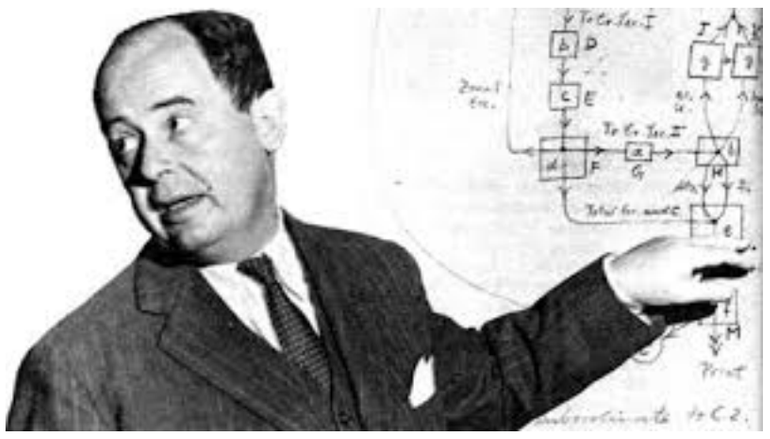


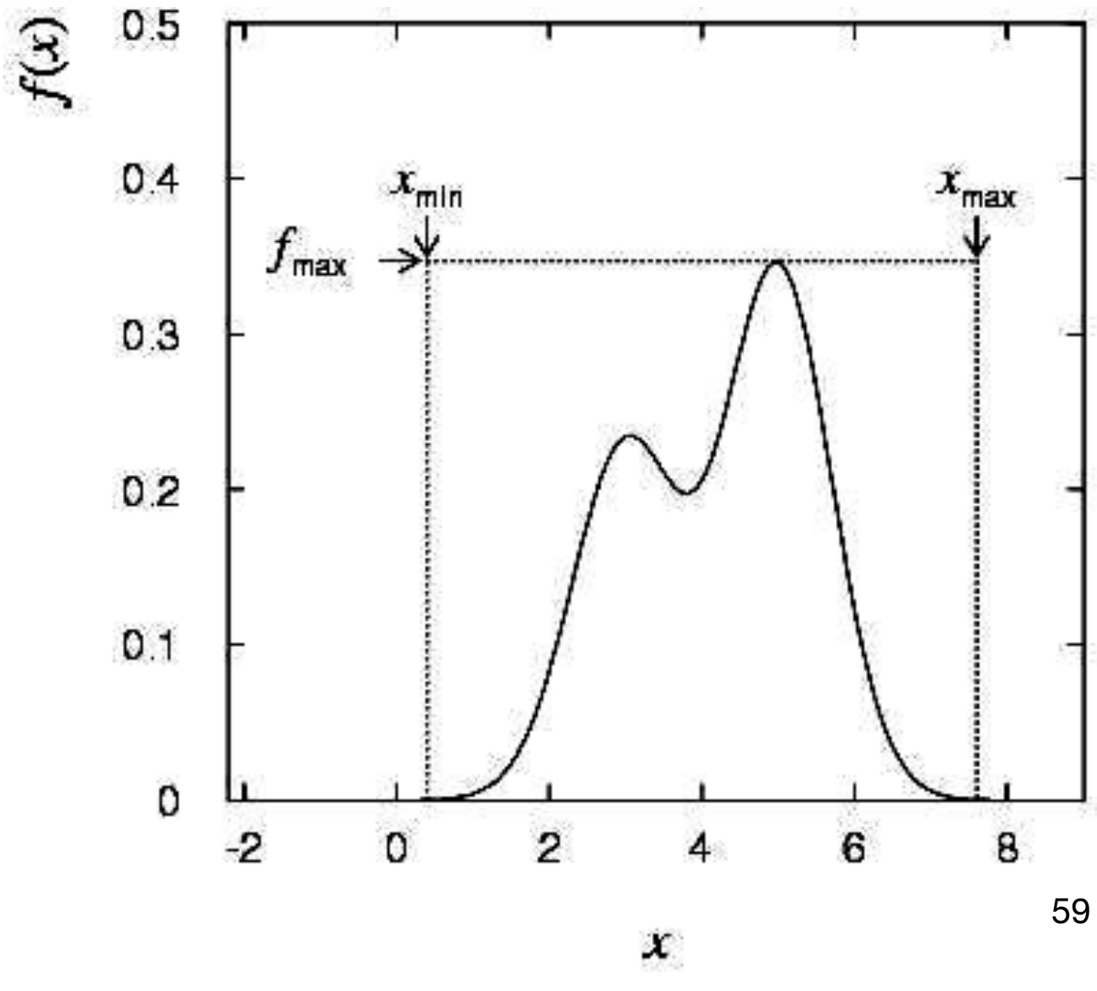
# From uniform to $p$(x): the (von Neumann) accept-reject method

- Enclose the $f(x)$ in a "box"
  - Generate $x_i$ uniformly in [$x_{min}$, $x_{max}$] starting from $r_i$ uniform in [0,1]

$$x_i = x_{min} + r_i\,(x_{max} - x_{min})$$

  - Generate $y_i$ uniformly in [0, $f_{max}$]
  - If $y_i < f(x)$, then accept $x_i$, otherwise reject it
- For improved efficiency smooth the "box" as much as possible



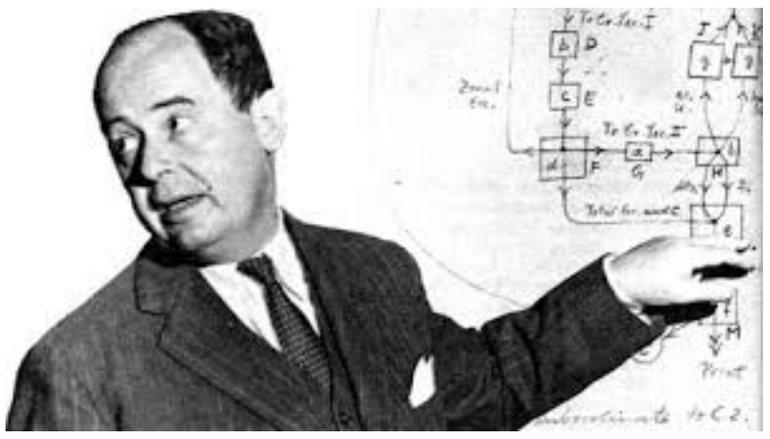


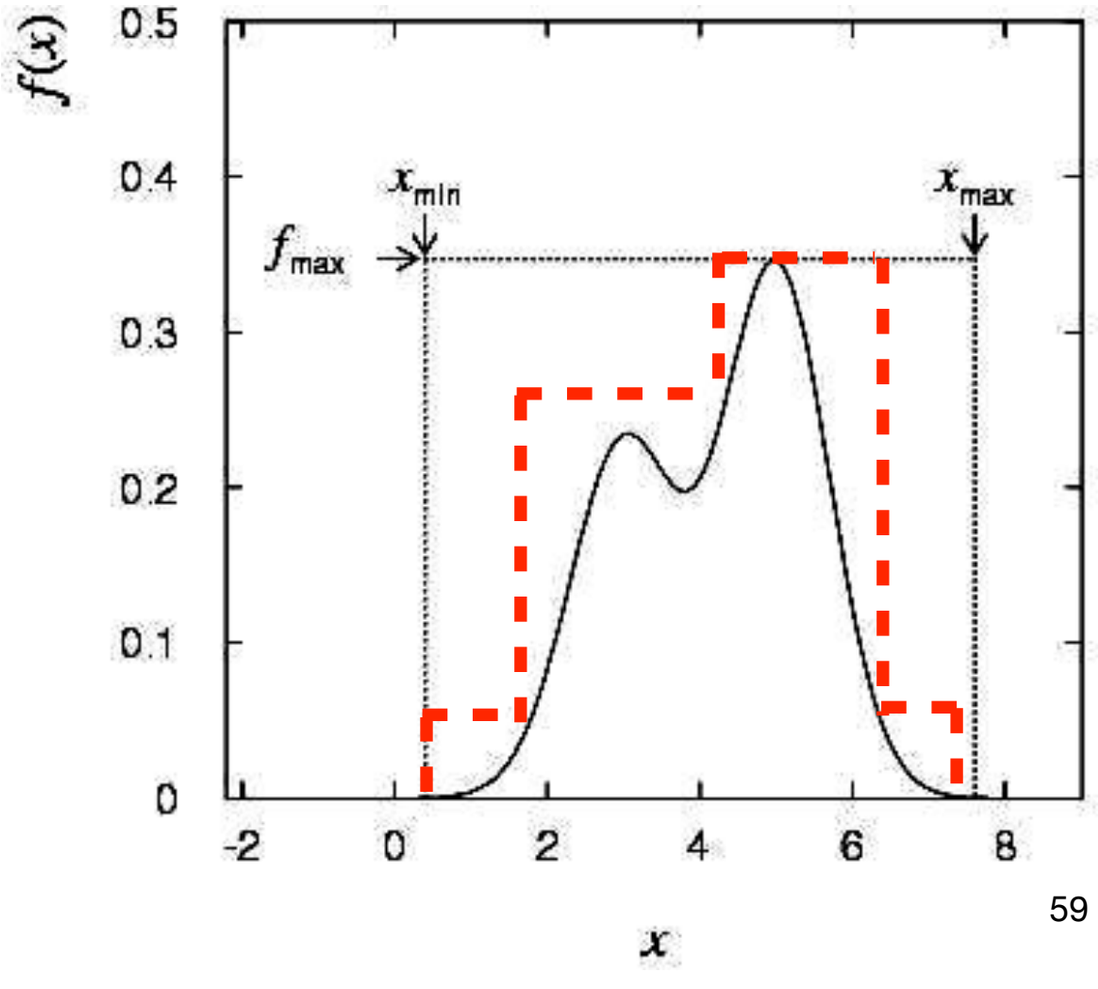
# From uniform to $p$(x): the (von Neumann) accept-reject method

- Enclose the $f(x)$ in a "box"
  - Generate $x_i$ uniformly in [$x_{min}$, $x_{max}$] starting from $r_i$ uniform in [0,1]

$$x_i = x_{min} + r_i\,(x_{max} - x_{min})$$

  - Generate $y_i$ uniformly in [0, $f_{max}$]
  - If $y_i < f(x)$, then accept $x_i$, otherwise reject it
- For improved efficiency smooth the "box" as much as possible





# Code to generate pseudodata

```
24 // Parameters used in the toy generation
25 const double toy_epssig(0.7), toy_epsbkg(0.3);//signal and background efficiencies
26 const double toy_fsig(0.4);//fraction of signal decays
27 const double toy_mu(1.865);//signal peak position
28 const double toy_spass(3.5e-3), toy_sfail(3.1e-3);//signal peak widths for candidates
      passing/failing the selection
29 const double toy_lbpass(1.5), toy_lbfail(16.);//background slopes for candidates passing/failing
      the selection
```

```
123     // Fill the input data
124     if ( args.gentoy ) {
125        std::cout << "Generating toy..." << std::endl;
126
127        TRandom3 r(args.seed);
128        unsigned int n = r.Poisson(args.nevents);
129        for (unsigned int i=0; i<n; i++) {
130           double m; bool pass;
131
132           if (r.Uniform(0.,1.) < toy_fsig) {
133              // generate signal event
134              do {
135                 // simulate signal efficiency
136                 pass = (r.Uniform(0.,1.) < toy_epssig);
137                 // generate mass
138                 m = (pass) ? r.Gaus(toy_mu,toy_spass) : r.Gaus(toy_mu,toy_sfail);
139              } while (m<m_min || m>m_max);
140           } else {
141              // generate bkg event with the iverse cumulative method
142              pass = (r.Uniform(0.,1.) < toy_epsbkg);
143              double l = (pass) ? toy_lbpass : toy_lbfail;
144              double u = r.Uniform(0.,1.);
145              m = -log(exp(-l*m_min) - u*(exp(-l*m_min)-exp(-l*m_max)))/l;
146           }
147
148           if (pass) hpass->Fill(m);
149           else hfail->Fill(m);
150        }
151     } else {
```

# Example 3: 2D fit with correlated variables

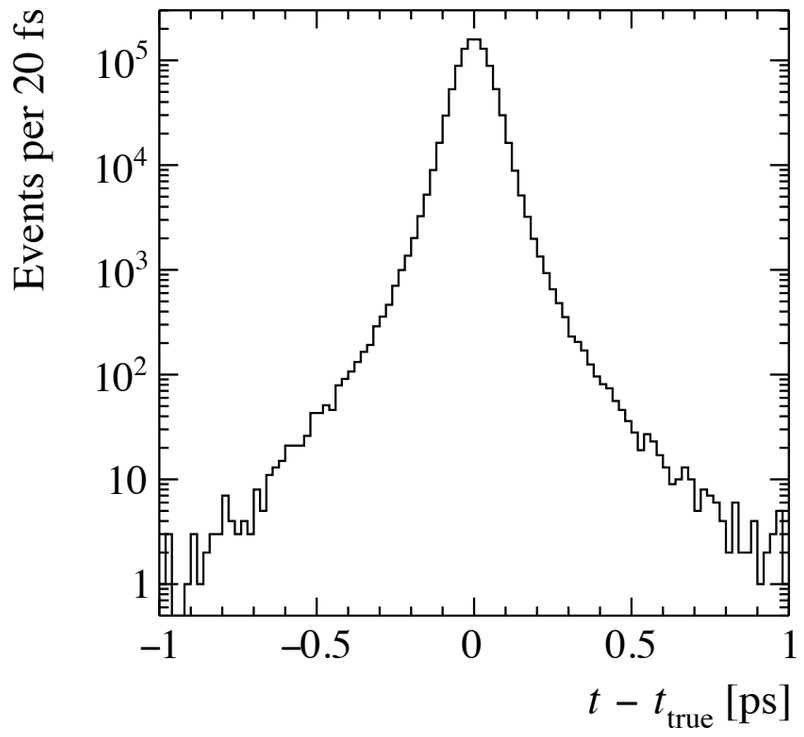
# Lifetime fit

- Assume you want to measure the lifetime of the $D^0$ meson by fitting the decay-time distribution

$$\mathrm{pdf}(t|\tau, b, \sigma, ...) \propto \int_0^\infty e^{-t_{\mathrm{true}}/\tau} R(t - t_{\mathrm{true}}|b, \sigma, ...) dt_{\mathrm{true}}$$

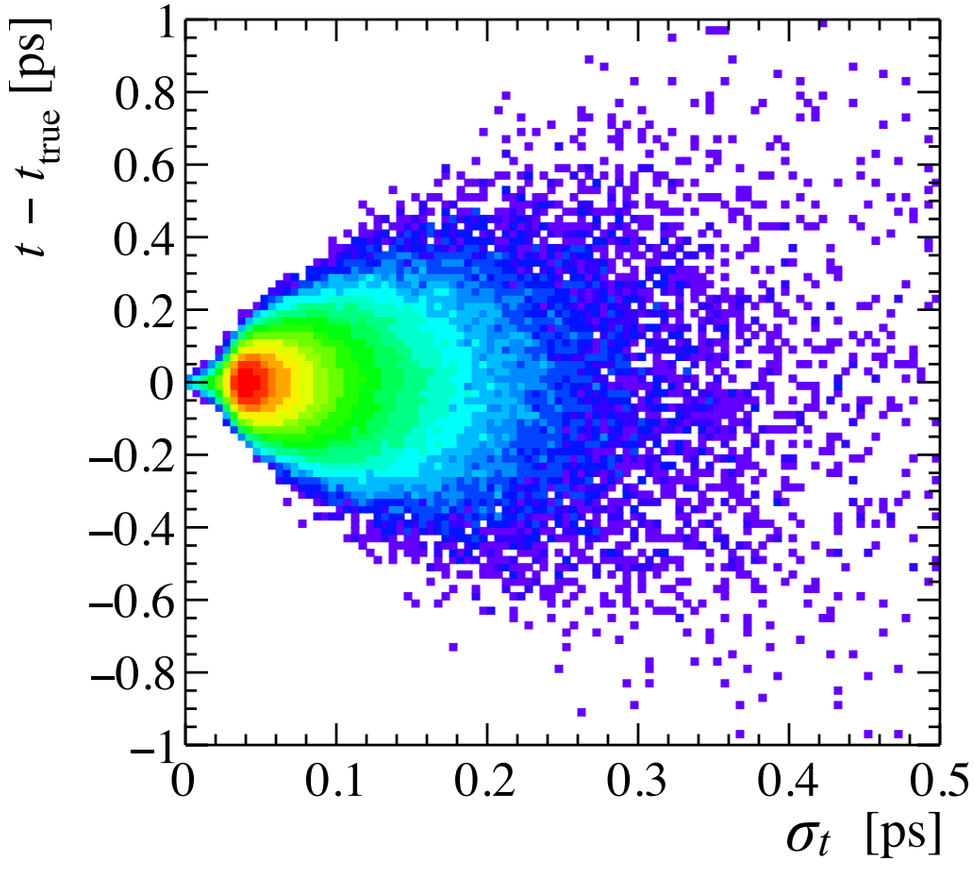
- The resolution function $R$ depends on the experimental setup and can be often quite complicated

- In general its mean value, $b$, quantifies a possible bias in the estimation of $t$ and its standard deviation, $\sigma$, quantifies how well the decay time is measured



# Lifetime fit with per-candidate resolution

- However, for some candidates the decay time is much easier to measure than for some others. The candidate-by-candidate decay-time uncertainty, $\sigma_t$, is then often used to have a more precise estimation of the resolution function

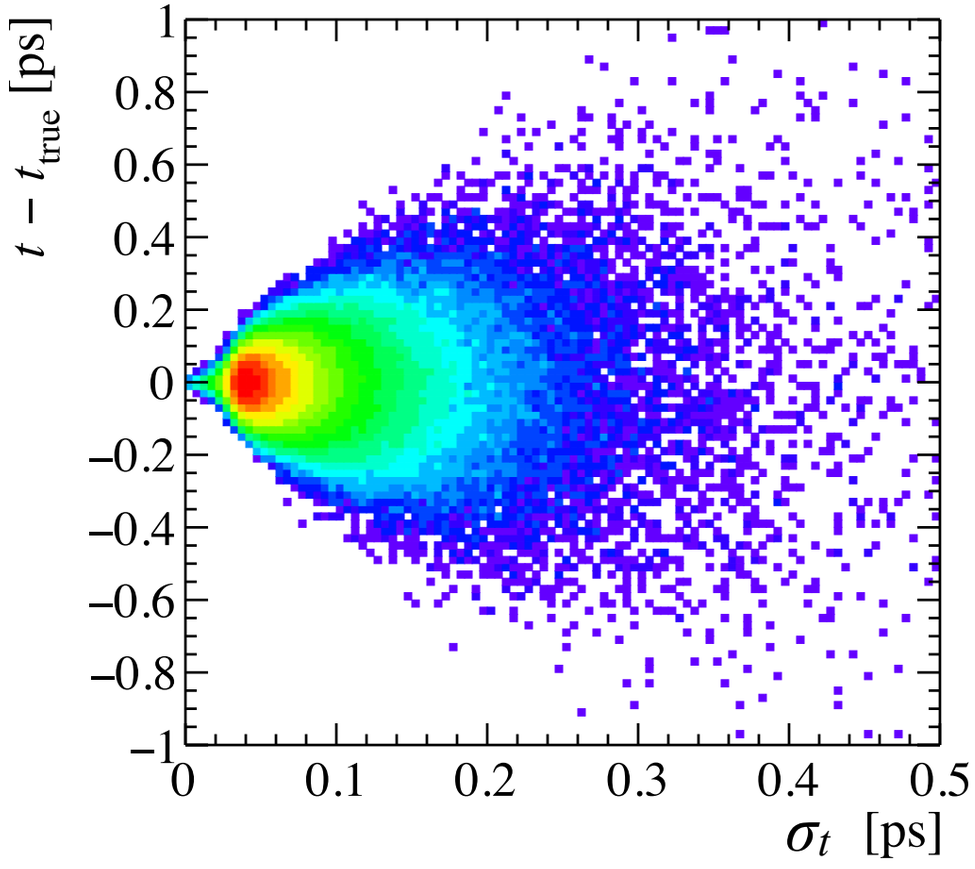



- The PDF now becomes function of two observables $t$ and $\sigma_t$, which in this case can be conveniently written as the product of two terms using the conditional probability:

$$\begin{aligned}\text{pdf}(t, \sigma_t|\tau, b) &= \text{pdf}(\sigma_t)\text{pdf}(t|\sigma_t, \tau, b) \\ &\propto \text{pdf}(\sigma_t) \int_0^\infty e^{-t_{\text{true}}/\tau} G(t - t_{\text{true}}|b, \sigma_t) dt_{\text{true}}\end{aligned}$$

# Lifetime fit with per-candidate resolution

- However, for some candidates the decay time is much easier to measure than for some others. The candidate-by-candidate decay-time uncertainty, $\sigma_t$, is then often used to have a more precise estimation of the resolution function




- The PDF now becomes function of two observables $t$ and $\sigma_t$, which in this case can be conveniently written as the product of two terms using the conditional probability:

This term is often forgotten/ignored

$$\text{pdf}(t, \sigma_t|\tau, b) = \text{pdf}(\sigma_t)\text{pdf}(t|\sigma_t, \tau, b)$$

$$\propto \text{pdf}(\sigma_t) \int_0^\infty e^{-t_{\text{true}}/\tau} G(t - t_{\text{true}}|b, \sigma_t) dt_{\text{true}}$$

# The Punzi effect

Giovanni Punzi:
``Comments on likelihood fits with variable resolution,''

*In the Proceedings of PHYSTAT2003: Statistical Problems in Particle Physics, Astrophysics, and Cosmology, Menlo Park, California, 8-11 Sep 2003, pp WELT002*

[arXiv:physics/0401045]

# A toy problem: sample composition

- Two classes of events, *A* and *B*, distinguished by variable *x*

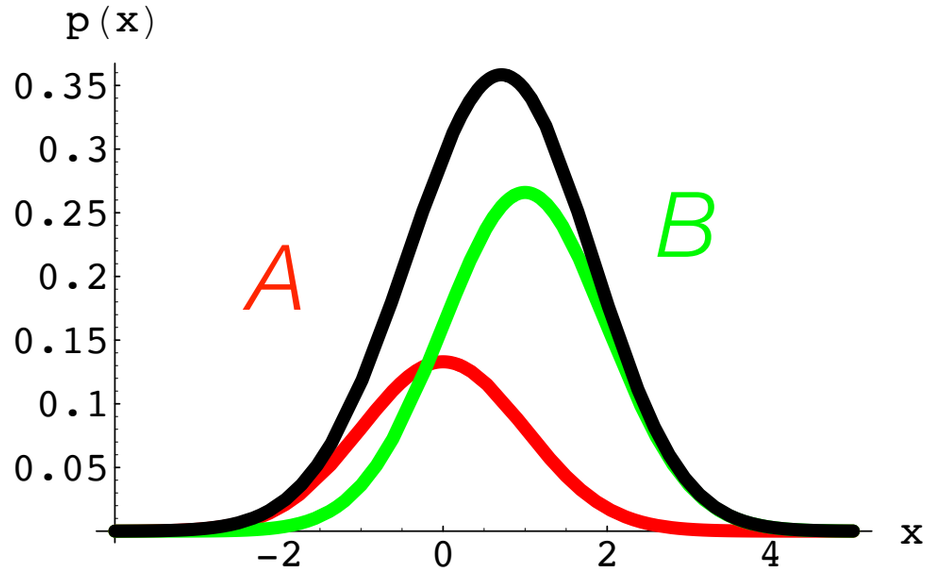
  $\mu(x) = 0, 1$ *for A, B*
  $\sigma(x)$ is unknown for both

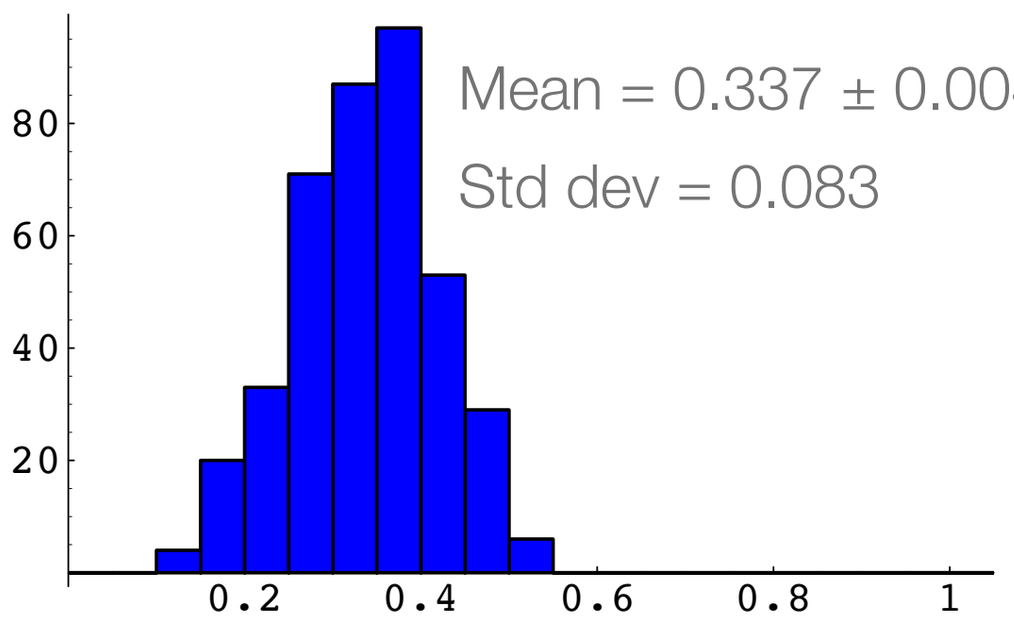
- Determine fraction of type-*A* events with likelihood fit:

$$L(f) = \prod_i [f\, G(x_i|\mu = 0, \sigma) + (1 - f)\, G(x_i|\mu = 1, \sigma)]$$

- Check result with toys generated with *f*=1/3

- All as expected. Statistical uncertainty on *f* is ~0.083



# Now a little modification

- Assume that for each event we measure, in addition to $x_i$, also its uncertainty $\sigma_i$

- We can use the per-event uncertainty $\sigma_i$ in place of the average uncertainty $\sigma$ to improve the precision of the estimated fraction (since we use more information)

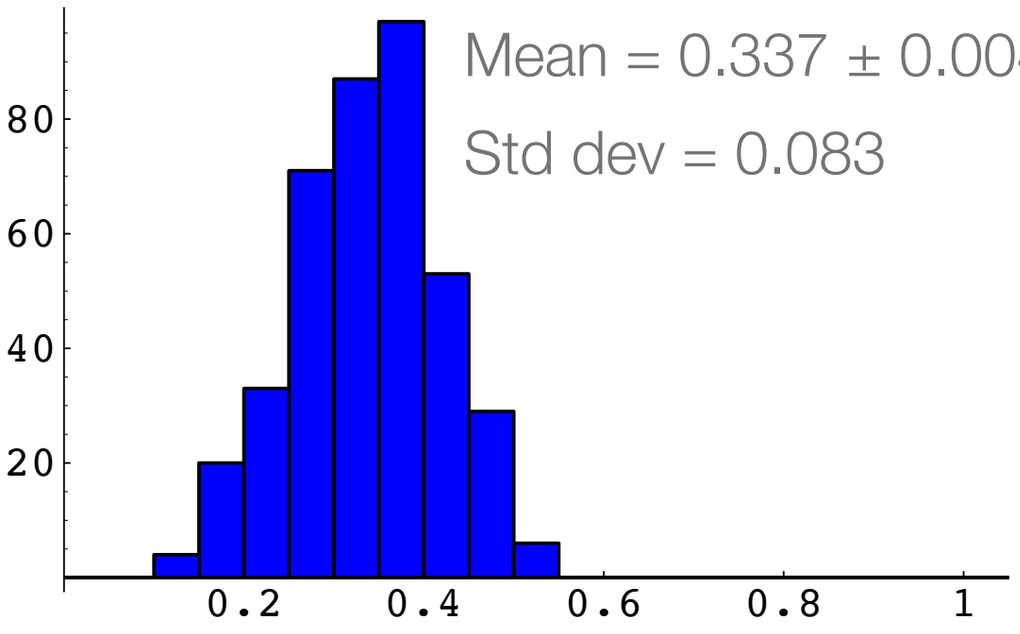
- One is tempted to write the new likelihood as

$$L(f) = \prod_i [f\, G(x_i|0,\sigma) + (1-f)\, G(x_i|1,\sigma)]$$

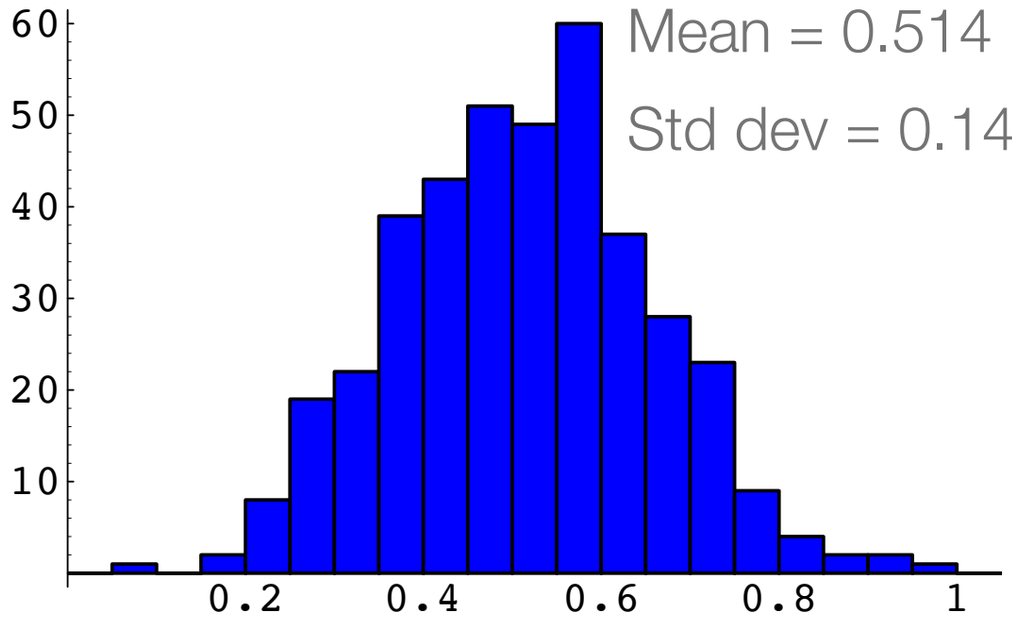
$$L(f) = \prod_i [f\, G(x_i|0,\sigma_i) + (1-f)\, G(x_i|1,\sigma_i)]$$

# Check again with toys and…

$$L(f) = \prod_i \left[ f\, G(x_i|0,\sigma) + (1-f)\, G(x_i|1,\sigma) \right]$$




$$L(f) = \prod_i \left[ f\, G(x_i|0,\sigma_i) + (1-f)\, G(x_i|1,\sigma_i) \right]$$




- *f* is biased and its uncertainty is larger. What did go wrong?
- NB: toy generated $\sigma_i$ uniformly distributed in [1.0,2.0] for category *A*, and in [1.5,3.0] for category *B*

# The likelihood is wrong

- The new problem is very different from the previous one: it has two observables ($x_i$,$\sigma_i$). The likelihood must now be written based on the PDF of the ($x_i$,$\sigma_i$) pair

- The expression

$$f\,G(x_i|0,\sigma_i) + (1-f)\,G(x_i|1,\sigma_i)$$

  is not the probability to find the pair ($x_i$,$\sigma_i$), $P(x_i,\sigma_i)$. Actually, it's not even the probability to find $x_i$, $P(x_i)$. It's the probability to find $x_i$ given $\sigma_i$, $P(x_i|\sigma_i)$

- It should be obvious that $P(x_i,\sigma_i)$ is not the same as $P(x_i|\sigma_i)$, nor the same as $P(x_i)$: *e.g.,*

$$P(\text{female}) \neq P(\text{female} \mid \text{pregnant}) \neq P(\text{female, pregnant})$$

# Fixing the mistake

- The (only) correct PDF is

$$\begin{aligned} P(x_i, \sigma_i) &= P(x_i, \sigma_i, (A \text{or} B)) \\ &= P(x_i, \sigma_i, A) + P(x_i, \sigma_i, B) \\ &= P(A)P(x_i, \sigma_i|A) + P(B)P(x_i, \sigma_i|B) \\ &= fP(x_i, \sigma_i|A) + (1-f)P(x_i, \sigma_i|B) \\ &= fP(x_i|\sigma_i, A)\cdot P(\sigma_i|A) + (1-f)P(x_i|\sigma_i, B)\cdot P(\sigma_i|B) \end{aligned}$$

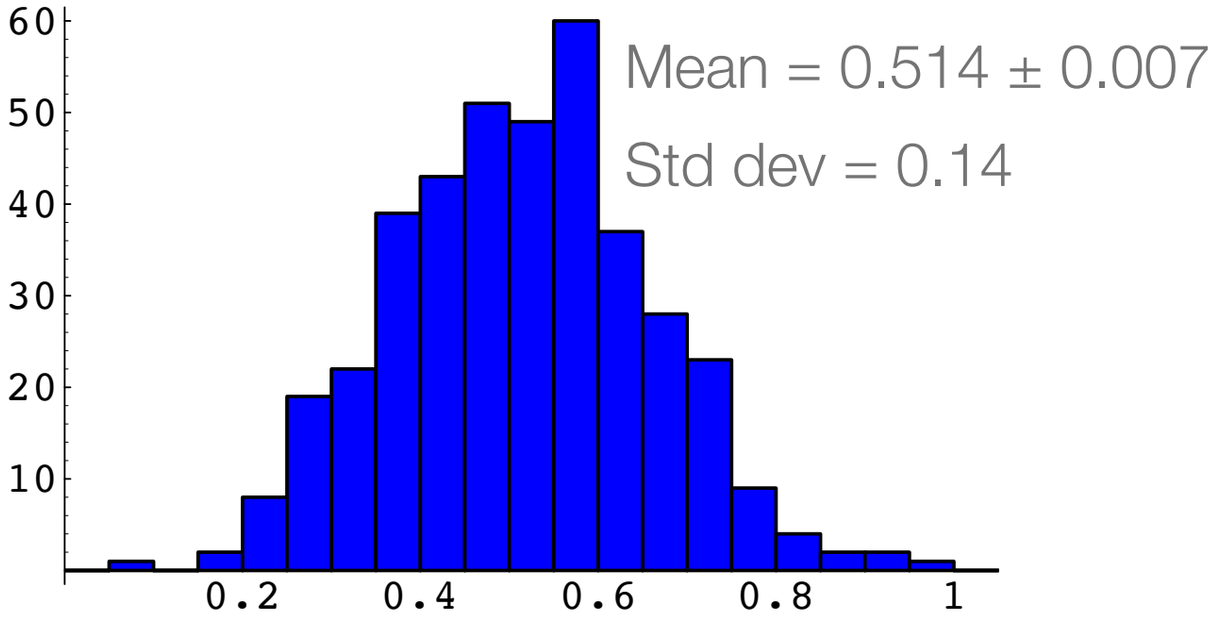
- Hence, the (only) correct likelihood is

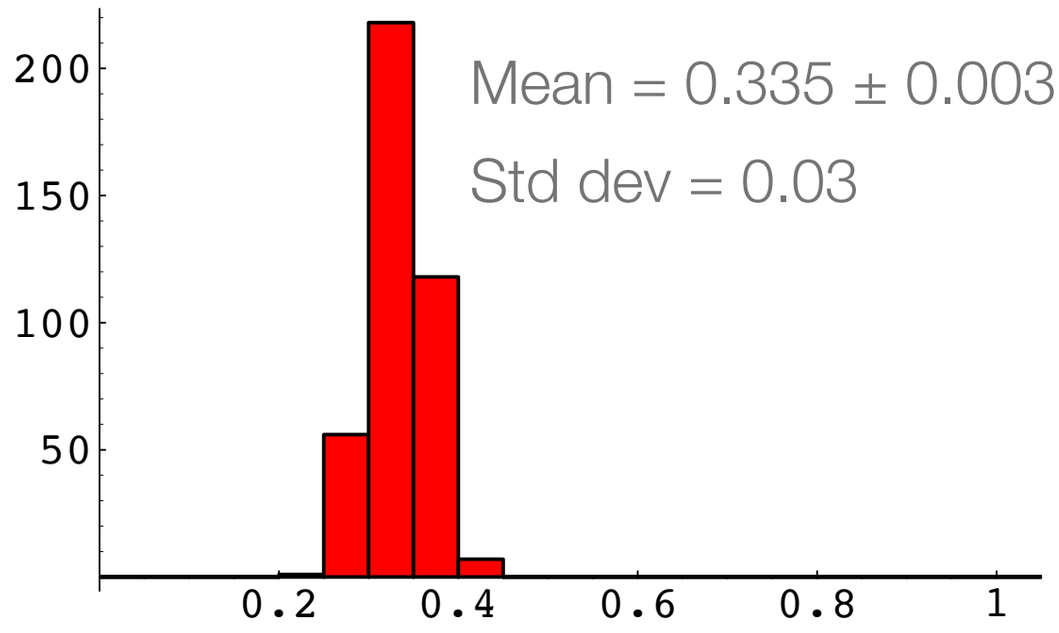
$$L(f) = \prod_i \left[f\, G(x_i|0, \sigma_i)P(\sigma_i|A) + (1-f)\, G(x_i|1, \sigma_i)P(\sigma_i|B)\right]$$

# Now it works!

$$L(f) = \prod_i \left[ f\, G(x_i|0,\sigma_i) + (1-f)\, G(x_i|1,\sigma_i) \right]$$

$$L(f) = \prod_i \left[ f\, G(x_i|0,\sigma_i) P(\sigma_i|A) + (1-f)\, G(x_i|1,\sigma_i) P(\sigma_i|B) \right]$$




- And, as expected, the uncertainty on the fraction is also smaller than in the case with fixed resolution

# Avoid the “Punzi effect”

- It may happen that the wrong likelihood actually works, but that’s not a good reason to use it (the PDF remains wrong!)
  - *E.g.*, IFF $P(\sigma_i|A) = P(\sigma_i|B)$, the term becomes an irrelevant constant multiplying the likelihood

$$L(f) = \prod_i \left[ f\, G(x_i|0,\sigma_i) + (1-f)\, G(x_i|1,\sigma_i) \right] P(\sigma_i)$$

$$\max L(f) = \max \prod_i \left[ f\, G(x_i|0,\sigma_i) + (1-f)\, G(x_i|1,\sigma_i) \right]$$

- Always explicitly assign a PDF for all observables (*i.e.*, any quantity with an event index *i*) to each of your likelihood terms

# Typical hiccups from real-life fitting

# The fit does not converge… there has to be something wrong…

- Not necessarily

- The convergence of a minimization procedure is a numerical issue associated with our lack of skill in finding suitable algorithms for minimizing complicated functions

- In principle, it should always be possible to find the minimum of a function evaluated over a set of data and as a function of some parameters

- Convergence or not convergence per se does not tell much about the model quality

# The fit does not converge… there has to be something wrong…

- However, always look for mistakes on your side starting from the implementation of the `FCN` in the code: *e.g.*, incorrect PDF normalization in the likelihood, ill-defined problem with unneeded free parameters, *etc*…

- Started too far from the solution. The `FCN` may have unphysical local minima, especially at infinity in some variables. Change starting values to avoid these regions, change parametrization, or add boundaries (but remember the caveats from few slides ago)

- The fit may converge even for ill-defined problem, always check that the error matrix is positive-definite at the minimum (if not, the estimated uncertainties are meaningless)

# Frequent “mistakes” affecting the converge of a fit

- A straight line has two parameters, but a linear probability distribution function has only one:

$$\left.\begin{array}{c} f(x|a,b) = a + bx \\ \displaystyle\int_{x_1}^{x_2} f(x|a,b) = 1 \end{array}\right\} \quad \Longrightarrow \quad p(x|q \equiv b/a) = \frac{1 + q\,x}{(x_2 - x_1) + \dfrac{q}{2}(x_2^2 - x_1^2)}$$

# Frequent “mistakes” affecting the converge of a fit

- A straight line has two parameters, but a linear probability distribution function has only one:

$$\left.\begin{array}{c} f(x|a,b) = a + bx \\ \displaystyle\int_{x_1}^{x_2} f(x|a,b) = 1 \end{array}\right\} \quad \Longrightarrow \quad p(x|q \equiv b/a) = \frac{1 + q\,x}{(x_2 - x_1) + \dfrac{q}{2}(x_2^2 - x_1^2)}$$

- When fitting for fractions use a robust parametrization:

$$p(x|f_1, f_2, f_3) = \sum_{i=1}^{3} f_i p_i(x) \quad \Longrightarrow \quad \sum_{i=1}^{3} f_i = 1$$

$$0 \leqslant q_i \leqslant 1$$

$$f_1 = q_1$$
$$f_2 = q_2$$
$$f_3 = (1 - q_1 - q_2)$$

# Frequent “mistakes” affecting the converge of a fit

- A straight line has two parameters, but a linear probability distribution function has only one:

$$\left.\begin{array}{c} f(x|a,b) = a + bx \\ \displaystyle\int_{x_1}^{x_2} f(x|a,b) = 1 \end{array}\right\} \quad \Longrightarrow \quad p(x|q \equiv b/a) = \frac{1 + q\,x}{(x_2 - x_1) + \dfrac{q}{2}(x_2^2 - x_1^2)}$$

- When fitting for fractions use a robust parametrization:

$$p(x|f_1, f_2, f_3) = \sum_{i=1}^{3} f_i p_i(x) \quad \Longrightarrow \quad \sum_{i=1}^{3} f_i = 1$$

$$0 \leqslant q_i \leqslant 1$$

$$\begin{aligned} f_1 &= q_1 \\ f_2 &= q_2 \\ f_3 &= (1 - q_1 - q_2) \end{aligned}$$

$$0 \leqslant q_i \leqslant 1$$

$$\begin{aligned} f_1 &= q_1 \\ f_2 &= (1 - q_1)q_2 \\ f_3 &= (1 - q_1)(1 - q_2) \end{aligned}$$


Any
Questions