2018

INTRODUCTION TO NUMERICAL ANALYSIS

Lecture 3-6: Modeling of Data: Parameter Estimation

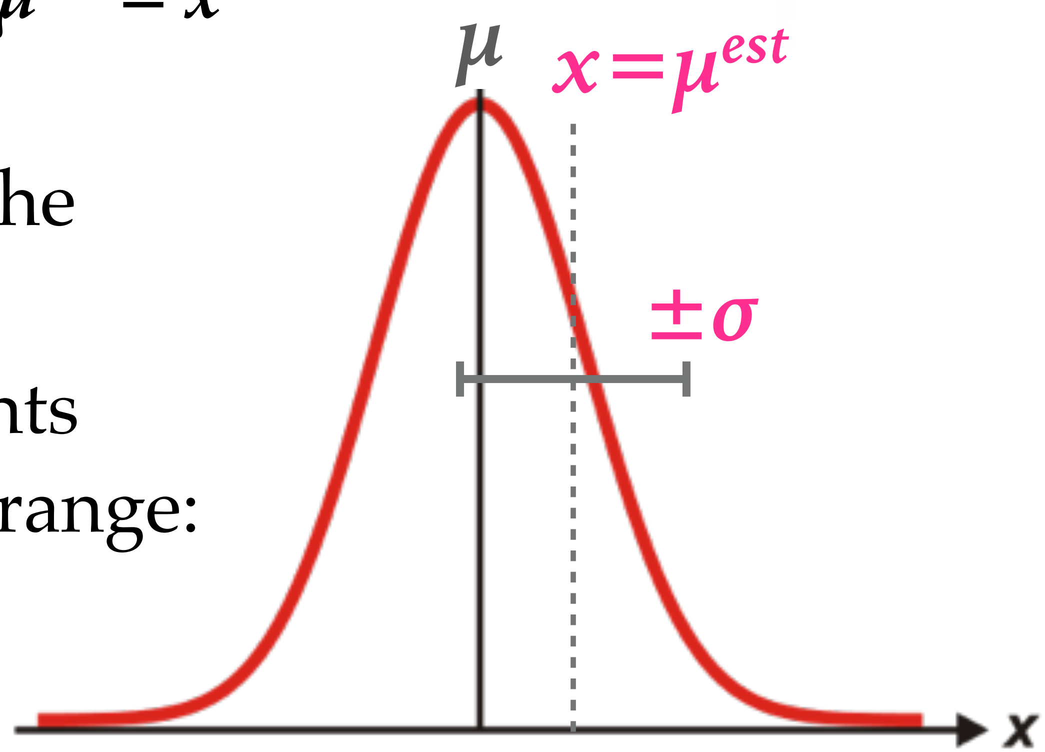
Kai-Feng Chen
National Taiwan University

THEORY OF ESTIMATORS

- Estimation may be considered as the measurement of a parameter (which is assumed to be fixed, but unknown value) based on a limited number of experimental observations.
  - **Point estimation**: determines a single value *as close as possible* to the true value — for example a measurement of physics parameter, such a mass, cross section, branching fraction.
  - **Interval estimation**: determines a range of values most likely to include the true parameter value — for example an estimation of upper/lower limits.
- The main subject here is what is the exact sense in which "close" and "likely to include"!

BASIC CONCEPTS

- To estimate a parameter, one first chooses a function of the observations = a method for proceeding from the observations to the estimate = **the estimator**.
- The numerical value yield by the estimator for a particular set of observations is the **estimate**.
- A minimal example — assuming we have a Gaussian PDF with a known σ and an unknown μ. A single experiment gives a measurement x, thus we estimate μ as $\mu^{est} = x$
  - The distribution of μ^{est} (*repeating the experiment many times*) should give the original Gaussian.
  - On average 68.27% of the experiments will provide an estimate within the range: $\mu-\sigma<\mu^{est}<\mu+\sigma$, thus $\mu = \mu^{est} \pm \sigma$.



THE LEAST SQUARES METHOD / CHI-SQUARE METHOD

- We have discussed already: consider a set of N observations of X_1, X_2, …, X_N, from a distribution with expectations of $\mathbf{E}(X_i, \theta)$ and the covariance matrix V. By **minimizing the covariance form**:

$$Q^2 = \sum_{i=1}^{N}\sum_{j=1}^{N}[X_i - E(X_i,\theta)](V^{-1})_{ij}[X_j - E(X_j,\theta)]$$
$$= [X - E(X,\theta)]^T V^{-1}[X - E(X,\theta)]$$

it provides an estimate of the unknown parameters .

- The covariance matrix V is not diagonal in general case. However if the observations are independent, the covariance matrix is diagonal. In this case the covariance form can be simplified to just sum of squares

$$Q^2 = \sum_{i=1}^{N}\frac{[X_i - E(X_i,\theta)]^2}{\sigma_i^2(\theta)} \quad \text{where } \sigma_i^2(\theta) = V_{ii}$$

THE MAXIMUM LIKELIHOOD METHOD

- Consider a set of N independent observations of ***X***: $X_1, X_2, \ldots, X_N$. They can be ***N*** events found in an experiment, and the joint PDF of ***X*** is

$$P(X|\theta) = P(X_1, X_2, \ldots, X_N|\theta) = \prod_{i=1}^{N} f(X_i|\theta)$$

where ***f(X,θ)*** is the PDF of any observation ***X***.

- When the variable ***X*** is replaced the observed data X^0, then ***P*** is no longer a PDF. It becomes the **likelihood function *L***, as a function of ***θ***:

$$L(\theta) = P(X|\theta)\Big|_{X=X^0}$$

- The maximum likelihood estimate of the parameter ***θ*** is that value for which **L has its maximum given the particular observation X^0**.

THE MAXIMUM LIKELIHOOD METHOD (2)

- In many cases it is convenient to take the logarithm, hence the production of probability can be converted to a summation:

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

- The "best fit" parameters can be obtained by maximizing the (*log*) likelihood function, or solving the likelihood equation as below:

$$\frac{\partial}{\partial\theta}\sum_{i=1}^{N} \ln f(X_i,\theta) = \frac{\partial}{\partial\theta} \ln L(X|\theta) = 0$$

- If the number of observations ***N*** is also a random variable, the **extended likelihood function** is can be introduced:

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

In the most common case p is a Poisson distribution

STILL ABOUT MINIMIZING

- Both the least squares estimator and maximum likelihood estimator requires minimizing or maximizing a function ("merit function!"):
  - For the least squares estimator, this can be carried out by simply supplying the corresponding Q^2 (χ^2) function.
  - For the ML estimator, it is common to supply **–2ln*L*** instead. The negative sign is required since common tools always does minimizing, and the factor of two will matches the supplied function as just sum of squares, **if the PDFs are all Gaussians**:

$$-2\ln L = \sum_{i=1}^{N} \frac{(X_i - \mu)^2}{\sigma^2} + \text{Const.}$$

- This operation can be performed with the SciPy tools introduced before, but we are going to use a *different one*.

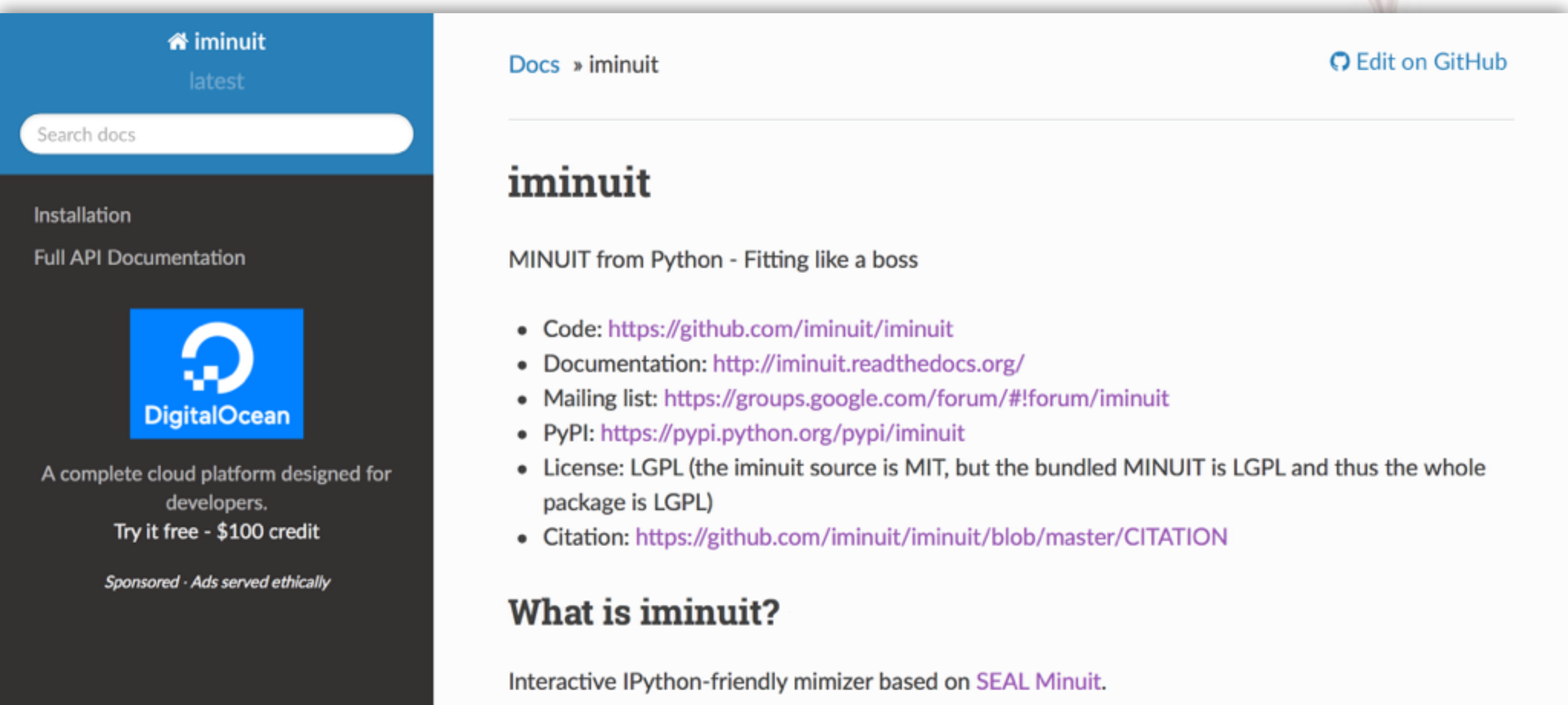
INTERFACE WITH MINUIT

- **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**, 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.
- Minuit is historically (*and still the case nowadays*) the most used minimization engine in particle physics.
  - It was a part of CERN software library (*written in fortran*), but it has been rewritten in C++.

INTERFACE WITH MINUIT (2)

- Well, we are using Python as our core language. Thus we will use a wrapper named **"iminuit"**:

http://iminuit.readthedocs.io/en/latest/



INSTALLATION OF IMINUIT

- If you are using anaconda package, this can be done by typing this under your terminal:

```
conda install iminuit
```

- If you are not using anaconda, the package can be installed though pip:

```
pip install iminuit
```

- A quick test can be made by just import the **iminuit** module directly:

```
% python
Python 3.6.8 |Anaconda, Inc.| (default, Dec 29 2018,
19:04:46). . . .
>>> import iminuit
>>>
```

STRUCTURE OF A MINUIT PROGRAM

- Let's perform a 2D minimum finding with the **iminut**:

```
from iminuit import Minuit

def fcn(x,y):
    return (x-8.)**2 + (y-6.)**2

m = Minuit(fcn, x=3., y=4.) ⇐ initial values: x=3, y=4!
m.migrad()

for par in m.values:
    print(par,":",m.values[par],"+-",m.errors[par])
```

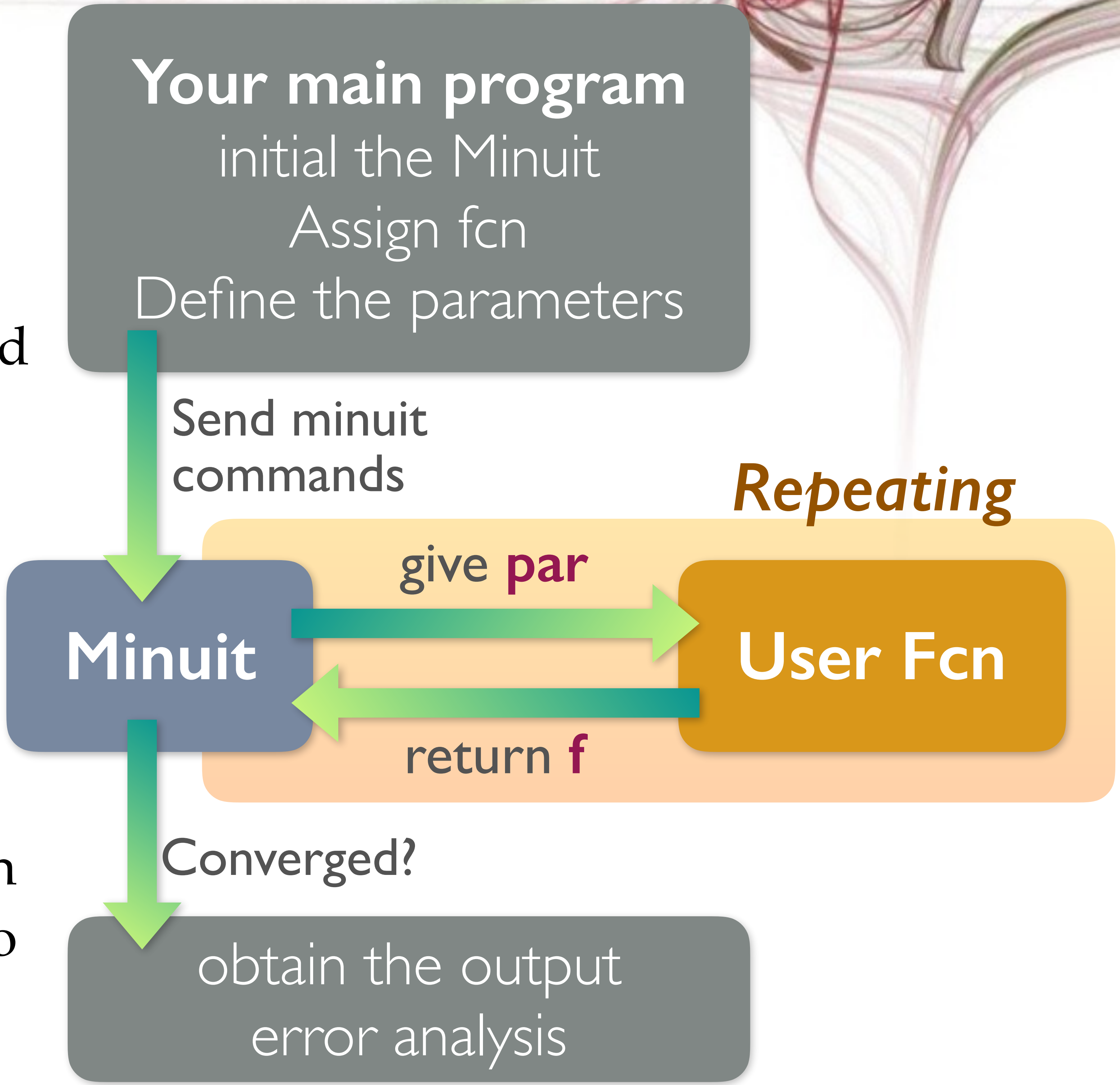
l306-example-01.py

- 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. This is not really different from the SciPy tool!

WORKFLOW

How a Minuit program runs:

- Your main program has to initialize the Minut 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.



- The corresponding terminal output:

```
**************************************************
*                     MIGRAD                     *
**************************************************

**********************************************************************
----------------------------------------------------------------------
fval = 1.6239748394623575e-22 | total call = 24 | ncalls = 24
edm = 1.6240799333014984e-22 (Goal: 1e-05) | up = 1.0
----------------------------------------------------------------------
|      Valid | Valid Param | Accurate Covar | Posdef |   Made Posdef |
----------------------------------------------------------------------
|       True |        True |           True |   True |         False |
----------------------------------------------------------------------
| Hesse Fail |     Has Cov |      Above EDM |        | Reach calllim |
----------------------------------------------------------------------
|      False |        True |          False |     '' |         False |
----------------------------------------------------------------------


---------------------------------------------------------------
|   | Name | Value | Para Err | Err- | Err+ | Limit- | Limit+ |
---------------------------------------------------------------
| 0 |    x =  8     |  1       |      |      |        |        |
| 1 |    y =  6     |  1       |      |      |        |        |
---------------------------------------------------------------

**********************************************************************
x : 8.000000000011832 +- 1.0000000000003002
y : 6.000000000004732 +- 0.9999999999993588
```

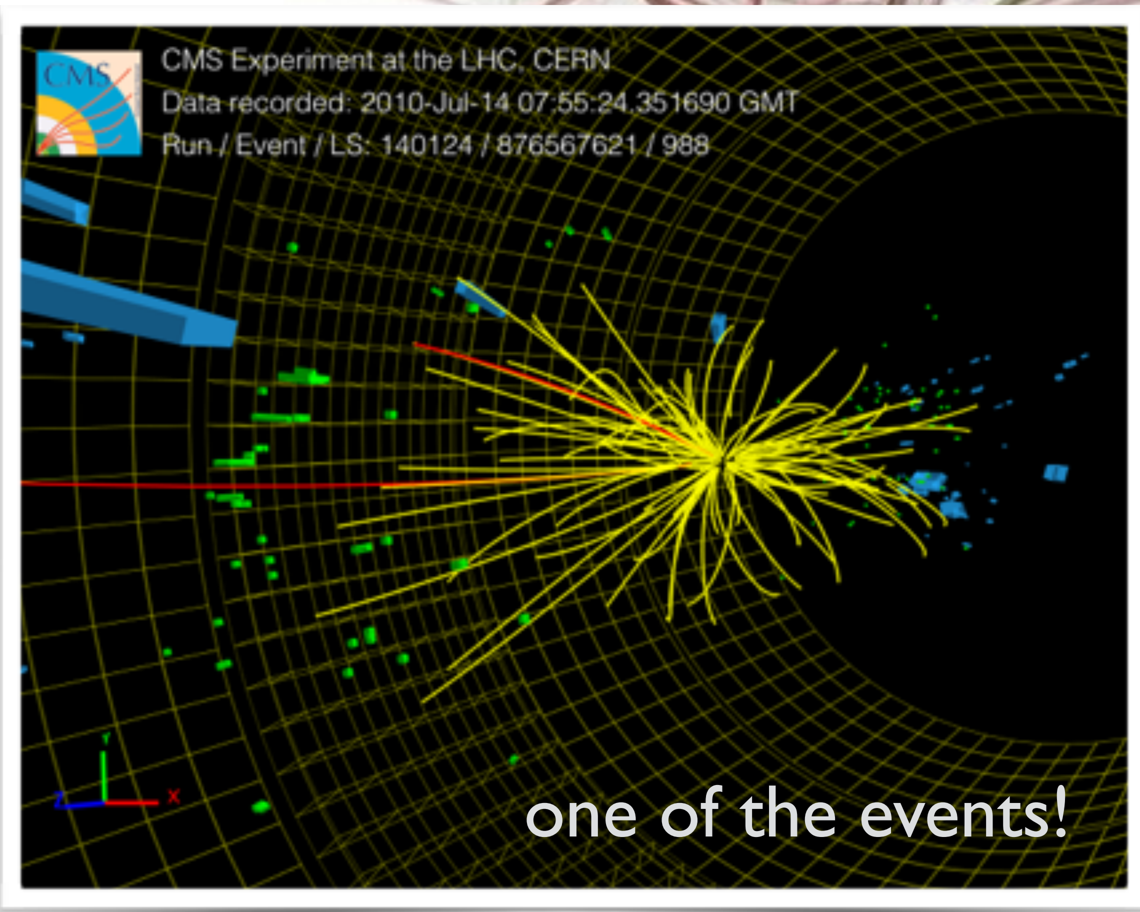
⇐ the best fit values are (8,6), w/ error (1, 1)!

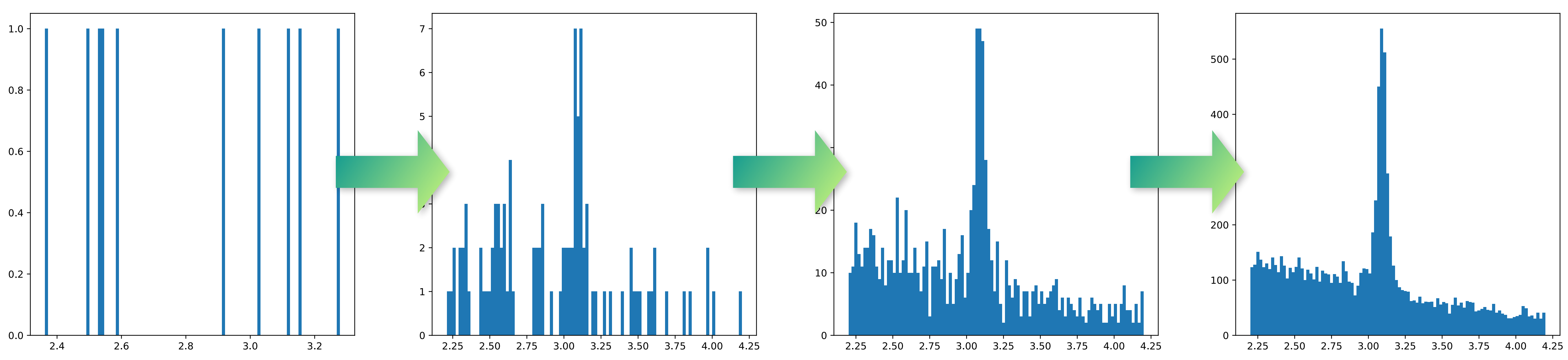
A TYPICAL EXAMPLE

- Suppose you are collecting a type of particle, the only observable is the *mass*. However there are several typical issues we have to resolve before reporting the measurement:
  - Our particle detector has some **finite resolution** hence the measured particle mass does not follow a delta function. The mass of the particle is not yet known, and the resolution of your detector is also unknown.
  - There are some random physics or detector noise with a unknown rate. This will generate some **fake background events** and mix with your signal.
  - Fortunately we can repeat such an experiment for many, many times and enable us to describe the data **statistically**.

A TYPICAL EXAMPLE (II)

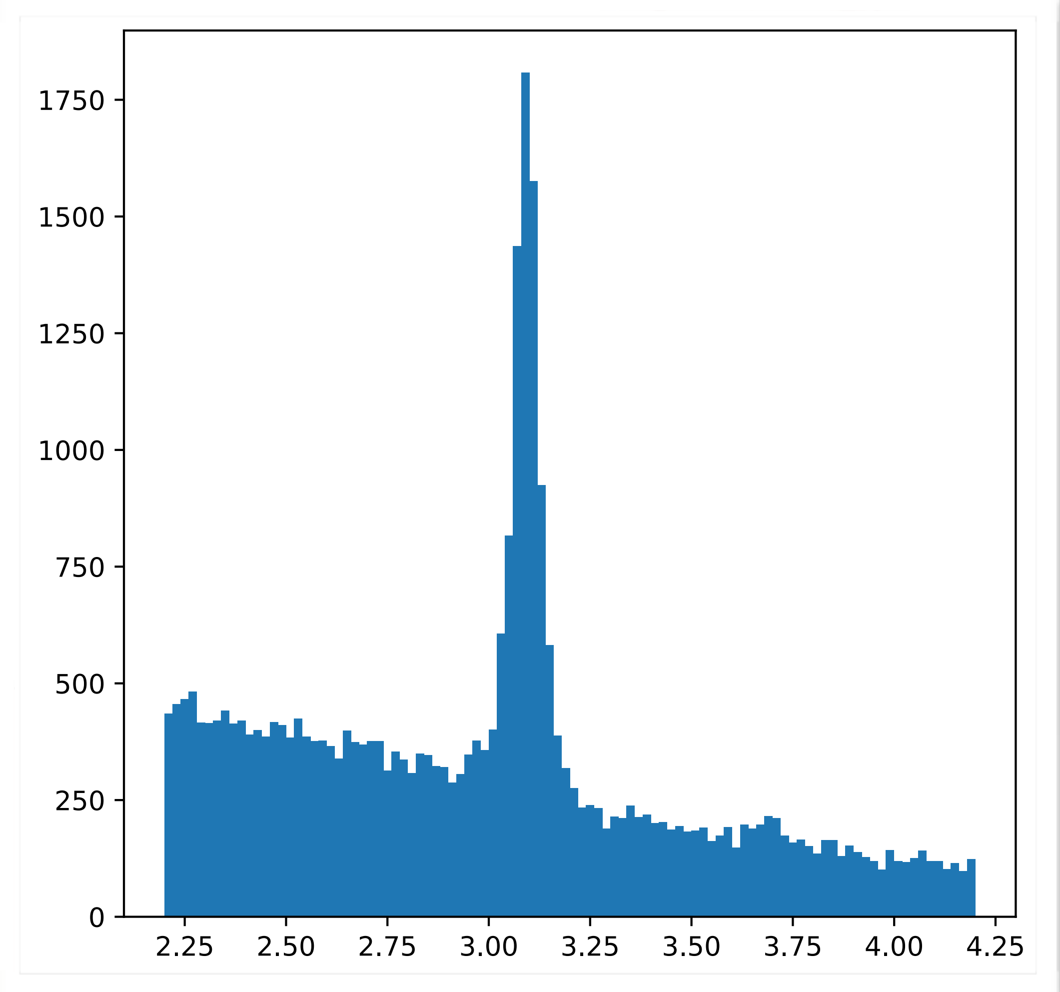
- This is very similar to what we have already introduced in the earlier lecture (when we are talking about the minimum finding!)
- But now the data is a given event-by-event, just like some random distribution generation!





A TYPICAL EXAMPLE (III)

- The data file `dimuon.npy` can be found on CIEBA and the lecture web page.
- The following example code can be used to produce the distribution show at the right.

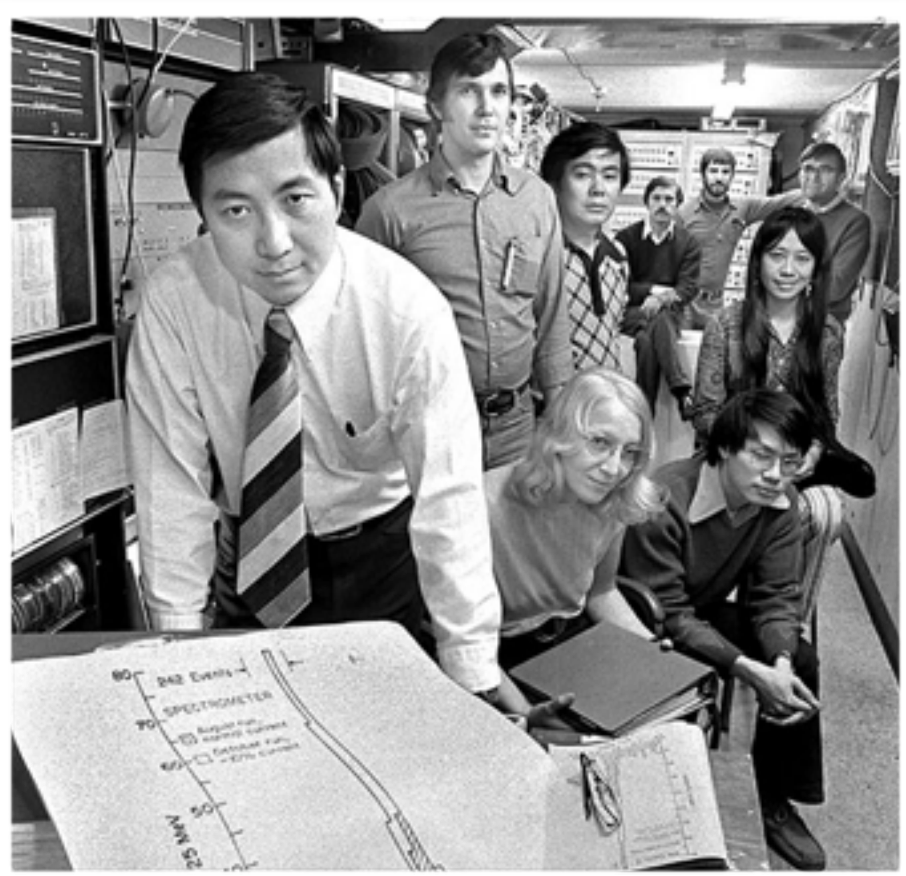


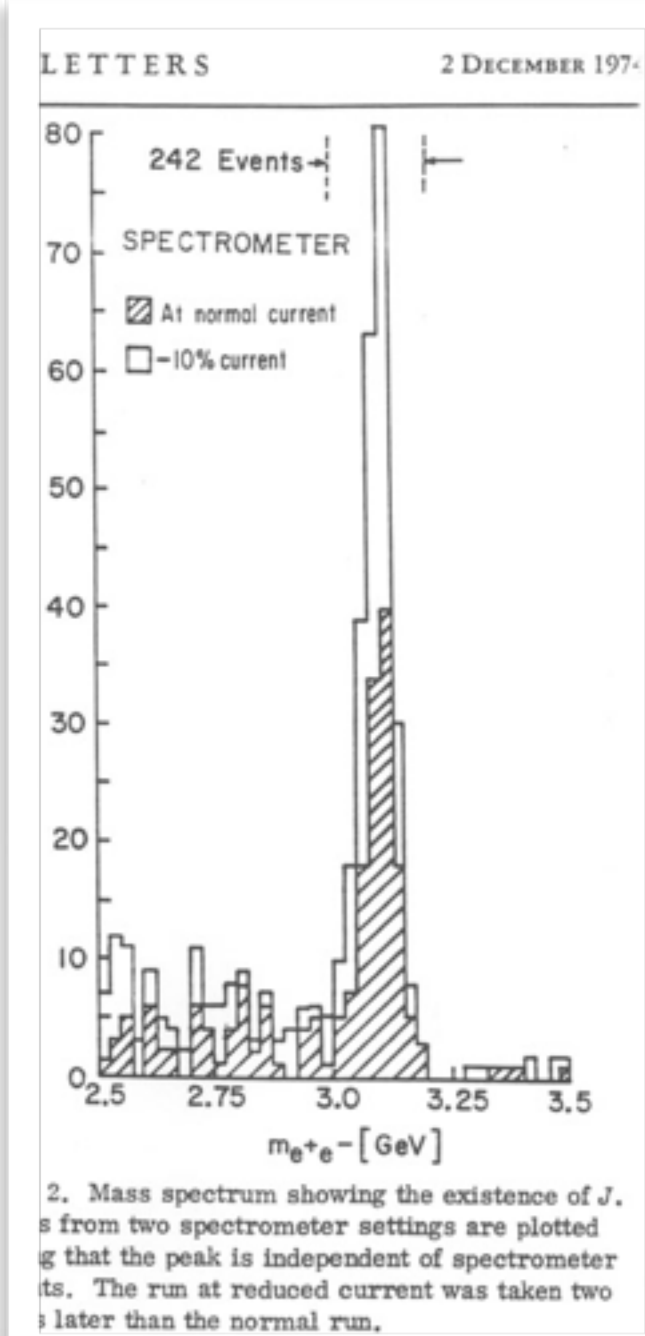

```python
import numpy as np
import matplotlib.pyplot as plt

evt = np.load('dimuon.npy')

fig = plt.figure(figsize=(6,6), dpi=80)
plt.hist(evt, bins=100, range=(2.2,4.2))
plt.show()
```

l306-example-02.py

A FAMOUS PARTICLE

- This "peak" you just saw is a famous one: the "J" particle (now it is called "J/ψ") found by Prof. Samuel Ting and other colleagues. This leads to his 1976 Nobel prize in Physics.
- It's a direct proof of the **charm quark**.

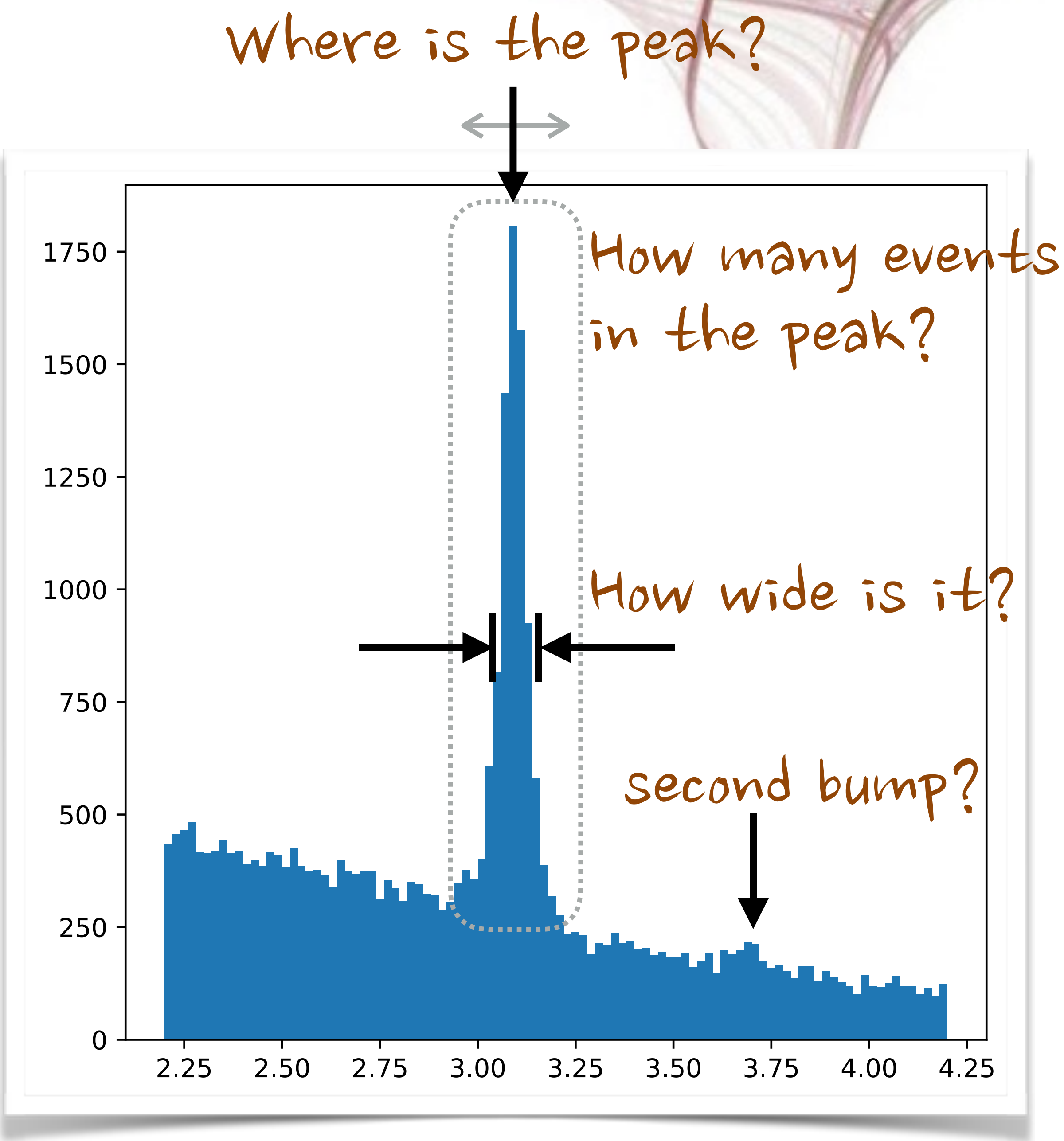




Surely today we have uncountable number of this particle, produced/detected by the modern experiments!

INFORMATION EXTRACTION?

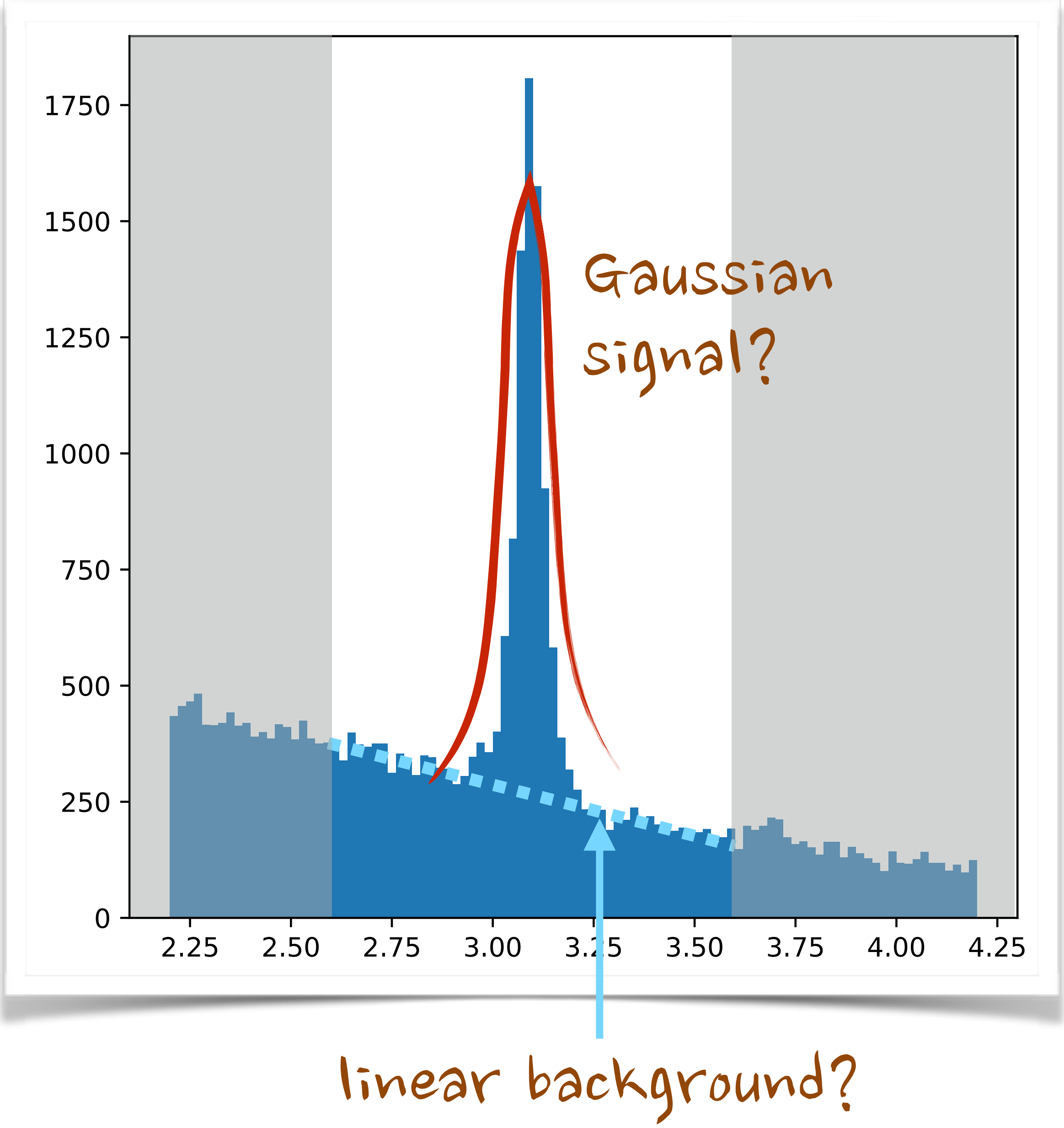
- Now we have data events summarized as a histogram, and several questions can be asked:
  - What is the peak position?
  - How many signal and background events we observed?
  - What is the width of the bump?
  - Is there a second peak nearby?
  - ……



By adopting a simple model to data, we can extract some of these information from the fits!

JUST MODEL IT?

- Let's take the events near the peak and describe it with a very simple model of **a Gaussian plus a linear function**. This is very close to the model used in our earlier lecture.
- You may ask how do we know if such a model is *sufficient* to fit the data events?
- **The answer is: we do not know!** But there are statistical methods can help you to decide which model describe your data better. You can look for **"f-test"**.



BINNED DATA

- In order to perform a **least-square/χ^2 fit**, first we have to convert the series of data into **"data points with uncertainties"**.
- Given the particle production process is mostly like a **Poisson**, the uncertainty can be just the variance of the distribution, ie. **the square-root of the event counts.**

Data
2.58406
3.11929
2.91791
2.49244
3.15518
⋮

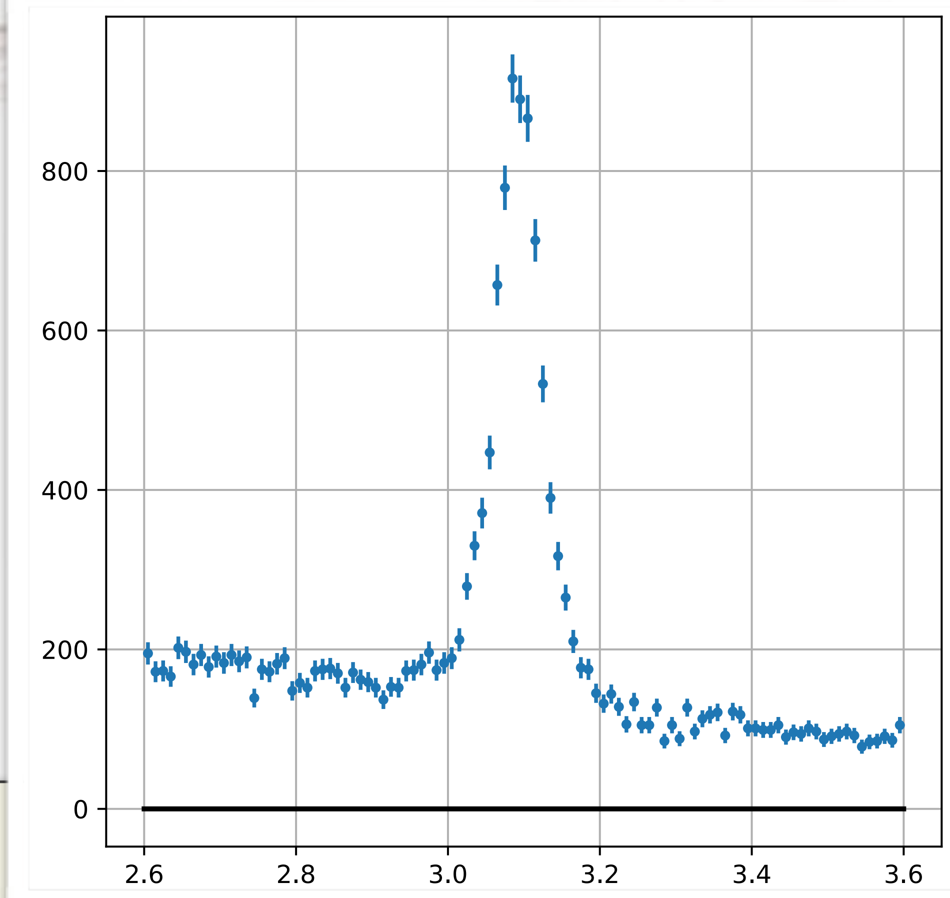
Count events in each bin

Bin	Histogram
2.60–2.61	195 ± 14
2.61–2.62	172 ± 13
2.62–2.63	173 ± 13
2.63–2.64	166 ± 13
2.64–2.65	202 ± 14
⋮	

Take square-root of the event counts as the uncertainty.

BINNED DATA (II)

- In fact we have practice in a much earlier lecture. NumPy can convert the data events to binned histograms.
- Then we can calculate the errors by ourselves easily.

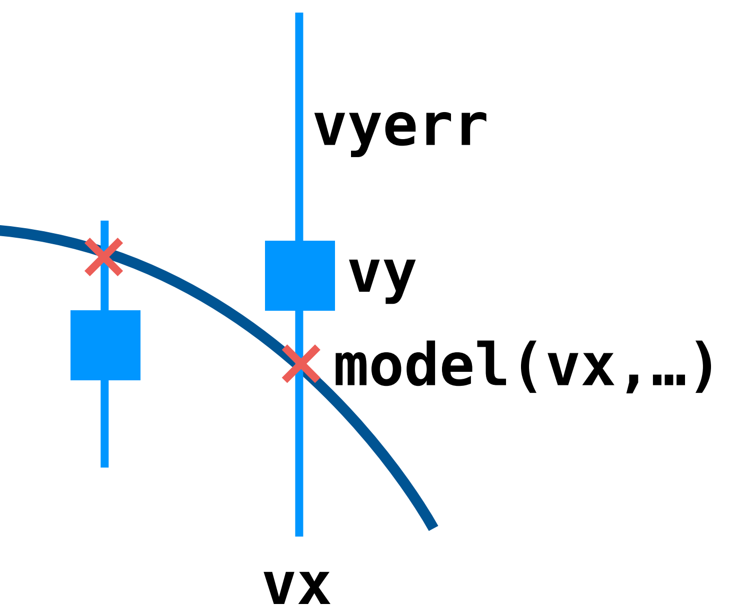



```
evt = np.load('dimuon.npy')

xmin, xmax, xbinwidth = 2.6, 3.6, 0.01
vy,edges = np.histogram(evt, bins=100, range=(xmin,xmax)) ⇐ y axis, x-edges
vx = 0.5*(edges[1:]+edges[:-1]) ⇐ x axis
vyerr = vy**0.5  ⇐ Poisson standard variance

fig = plt.figure(figsize=(6,6), dpi=80)
plt.plot([xmin,xmax],[0.,0.],c='black',lw=2)
plt.errorbar(vx, vy, yerr = vyerr, fmt = '.')
plt.grid()
plt.show()
```

l306-example-02a.py (partial)

LEAST-SQUARE FIT WITH MINUIT

- It is straightforward to perform a least-square fit with Minuit. Basically we have to provide a fcn function to evaluate the χ^2 value as we did before with SciPy.



$$\chi^2 = \sum \left(\frac{\mathrm{model}(vx; \ldots) - vy}{vyerr} \right)^2$$

```
def model(x, norm, mean, sigma, c0, c1):

    linear = c0 + c1*(x-xmin)/(xmax-xmin)

    gaussian = norm*xbinwidth/(2.*np.pi)**0.5/sigma * \
        np.exp(-0.5*((x-mean)/sigma)**2)

    return gaussian + linear

def fcn(norm, mean, sigma, c0, c1):
    expt = model(vx, norm, mean, sigma, c0, c1)
    delta = (vy-expt)/vyerr
    return (delta[vy>0.]**2).sum()
```

l306-example-03.py (partial)

LEAST-SQUARE FIT WITH MINUIT (II)

- Calling the Minuit to do the minimization, and overlay the resulting curves:

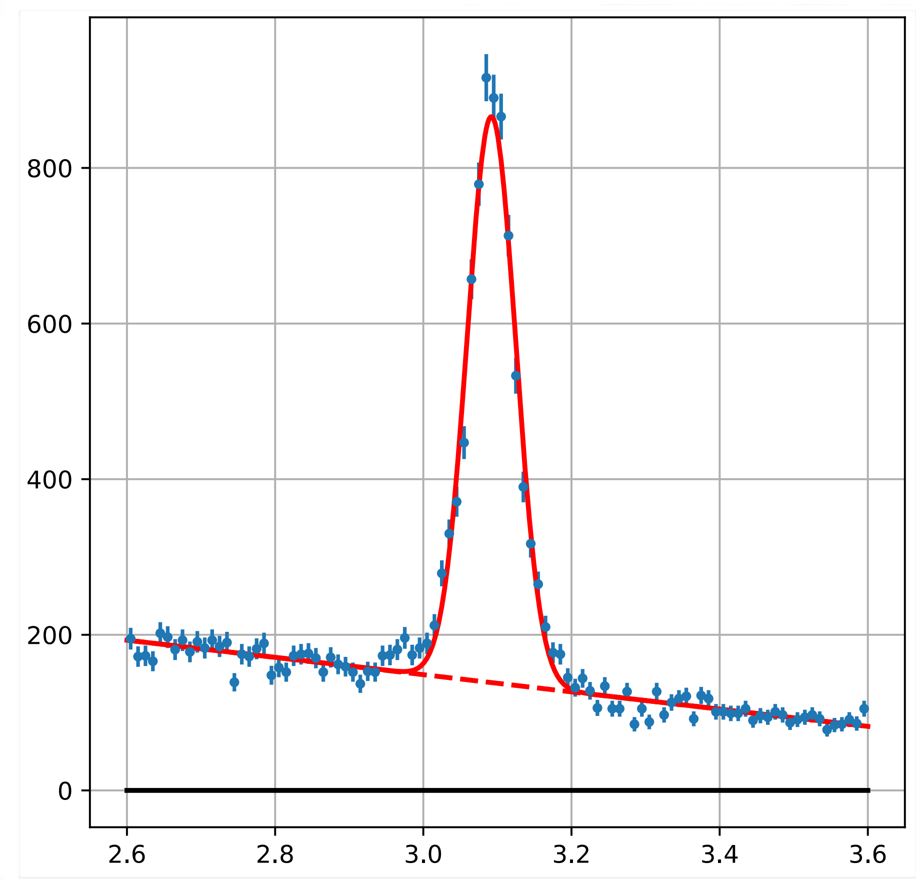
```
m = Minuit(fcn, norm=6000., mean=3.09, sigma=0.04, c0=200., c1=0.)
m.migrad() ⇐ Look for minimal
m.minos() ⇐ Calculate the asymmetric errors
m.print_param() ⇐ Print parameter summary

. . . . . .

plt.plot([xmin,xmax],[0.,0.],c='black',lw=2)
plt.errorbar(vx, vy, yerr = vyerr, fmt = ‘.')

cx = np.linspace(xmin,xmax,500)
cy = model(cx,m.values[‘norm'],m.values['mean'],
              m.values['sigma'],m.values['c0'],m.values['c1'])
cy_bkg = model(cx,0.,m.values[‘mean'],
                  m.values['sigma'],m.values['c0'],m.values['c1'])
plt.plot(cx, cy, c='red',lw=2)
plt.plot(cx, cy_bkg, c=‘red',lw=2,ls='--')

. . . . . .
```

Curve plotting is roughly ⇐ the same as before!

l306-example-03.py (partial)

LEAST-SQUARE FIT WITH MINUIT (III)

- We can obtain the best fitted values with their associated uncertainties!
  - We have observed 5984 ± 96 events.
  - The mean peak position is 3.0920 ± 0.6 GeV.
  - We will come back to the meaning of these uncertainties in a moment.



	Name	Value	Para Err	Err−	Err+
0	norm =	5984	96.34	−96.43	96.26
1	mean =	3.092	0.0005636	−0.0005661	0.0005615
2	sigma =	0.03282	0.0006026	−0.0005941	0.0006114
3	c0 =	193.1	2.668	−2.671	2.666
4	c1 =	−110.8	4.007	−4.002	4.013

THE LIMITATION OF LEAST-SQUARE METHOD

- The results shown in the previous page look quite nice, but there are some obvious problems! Remember we always need to produce a **histogram** before applying the chi-square fit to the data.
  - **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 very 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:

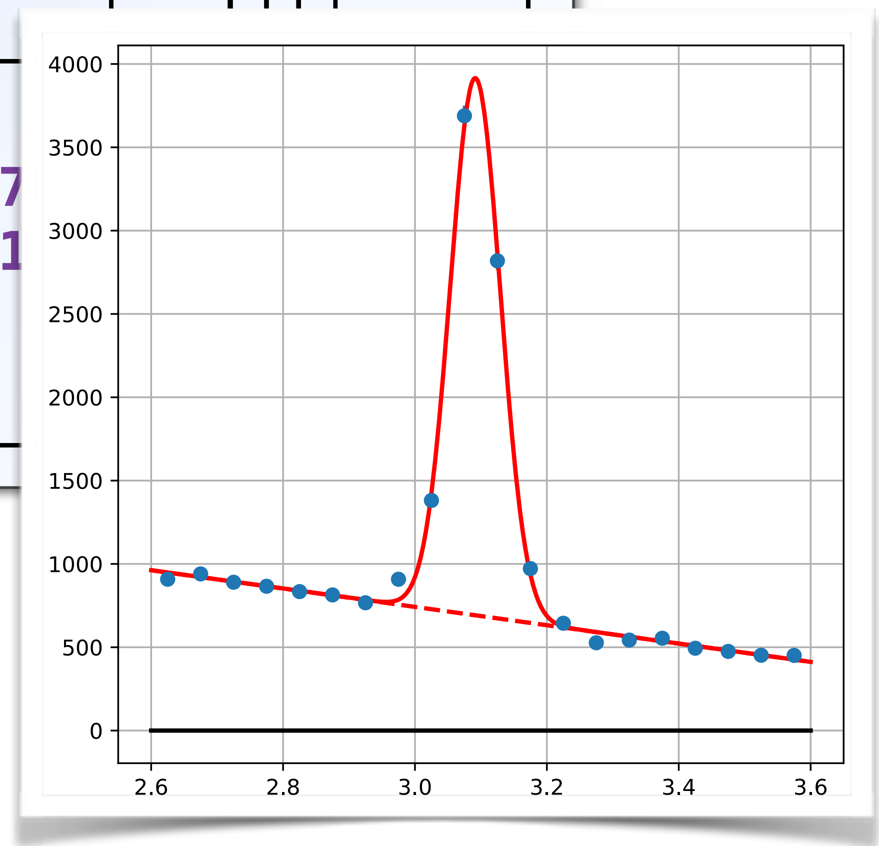
```
xmin, xmax, xbinwidth = 2.6, 3.6, 0.05
vy,edges = np.histogram(evt, bins=20, range=(xmin,xmax))
```

1306-example-03a.py (partial)

```
------------------------------------------------------------------------
|      | Name  |  Value     | Para Err     |    Err-       |    Err+     |
-----------------------------------------------------------
|    0 |  norm =  6151      |  100.2       | -100.3
|    1 |  mean =  3.092     |  0.0006676   | -0.000667
|    2 | sigma =  0.03806   |  0.000643    | -0.000641
|    3 |    c0 =  962.2     |  13.48       | -13.48
|    4 |    c1 = -550.1     |  20.15       | -20.16
-----------------------------------------------------------
```

```
|    0 |  norm =  5984      |
|    1 |  mean =  3.092     |
|    2 | sigma =  0.03282   |
```

Original fits



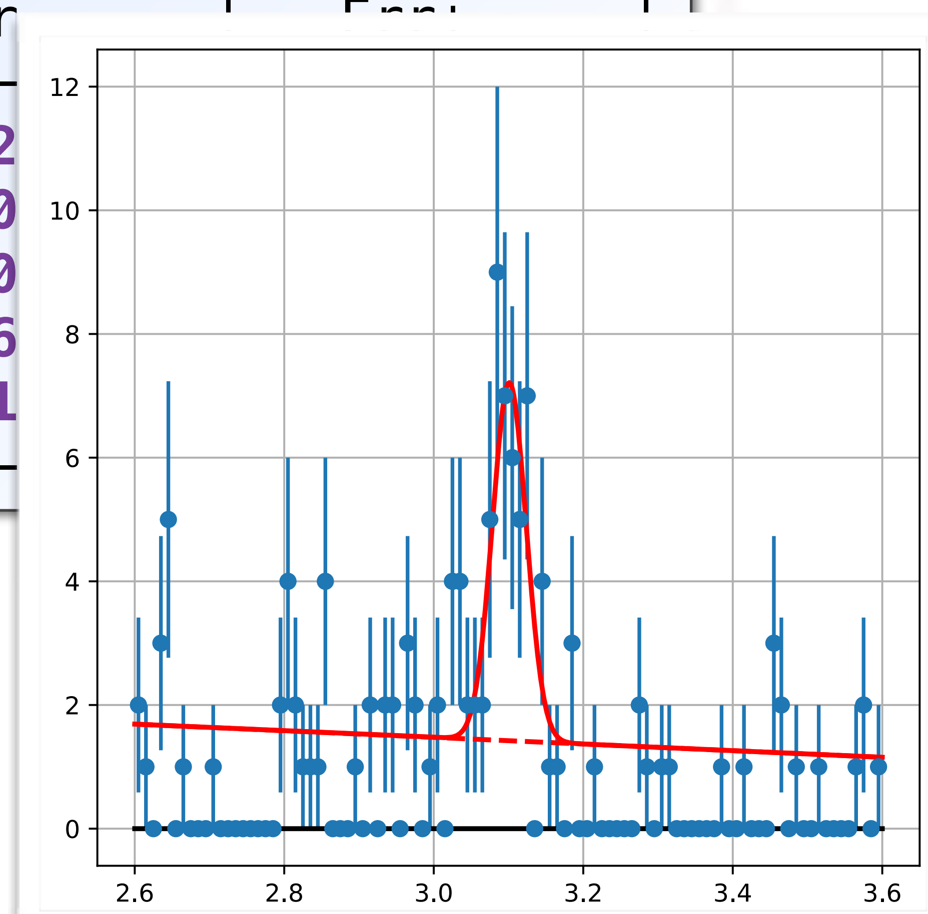
TRIAL #2: A MUCH SMALLER SAMPLE?

- Let's re-do the fit with only first **200** events?

```python
evt = np.load('dimuon.npy')[:200]
```

l306-example-03b.py (partial)

```
---------------------------------------------------------------------
|      | Name  |   Value   | Para Err  |    Err
---------------------------------------------------------------------
|   0 |  norm =  32.79    |  7.847    | -7.92
|   1 |  mean =  3.101    | 0.006163  | -0.00
|   2 | sigma =  0.02259  | 0.004775  | -0.00
|   3 |    c0 =  1.691    | 0.3638    | -0.36
|   4 |    c1 = -0.5362   | 0.6183    | -0.61
---------------------------------------------------------------------
```



The background level is totally overestimated:

FIT WITH UNBINNED MAXIMUM LIKELIHOOD

- 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:

$$L_i = f_s \cdot P_s(x_i; \mu, \sigma) + (1 - f_s) \cdot P_b(x_i; c_1)$$

The best solution by maximizing the total likelihood:

$$L = \prod_i^N L_i$$

Or, by minimizing the value of

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

Remember the factor of 2 is to match the definition of Gaussian errors!

P_s (P_b) : signal (background) PDF

f_s ($1-f_s$) : signal (background) fraction

μ, σ, c_1 : fitting parameters to be resolved by the estimator

FIT WITH UNBINNED MAXIMUM LIKELIHOOD (II)

- Now to prepare the explicit functions based on the simple model we just introduced earlier:

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

$$P_b = c_0 + c_1\cdot x = N[1 + c_1\cdot x]$$

The normalization is very important!

$$\int_{\min}^{\max} P_b(x)dx = 1 \rightarrow N = \frac{1}{(\max-\min) + c_1\cdot(\max^2-\min^2)/2}$$

$$L_i = f_s \times \frac{1}{\sqrt{2\pi}\sigma}\exp\left[\frac{-(x_i-\mu)^2}{2\sigma^2}\right] + (1-f_s)\times N\,(1+c_1\cdot x)$$

→ With the following floated fitting parameters: f_s, σ, μ, c_1

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

UML FITTER EXAMPLE

- Let's modify the code to perform the likelihood calculation:

```
evt = np.load('dimuon.npy')
evt = evt[abs(evt-3.1)<0.5]

xmin, xmax, xbinwidth = 2.6, 3.6, 0.01
vy,edges = np.histogram(evt, bins=100, range=(xmin,xmax))
vx = 0.5*(edges[1:]+edges[:-1])
vyerr = vy**0.5

def model(x, norm, mean, sigma, c1):

    linear = (1. + c1*x)/((xmax-xmin) + c1*(xmax**2-xmin**2)/2.)

    gaussian = 1./(2.*np.pi)**0.5/sigma * \
        np.exp(-0.5*((x-mean)/sigma)**2)
    fs = norm/len(evt)
    return fs*gaussian + (1.-fs)*linear

def fcn(norm, mean, sigma, c1):

    L = model(evt, norm, mean, sigma, c1)
    if np.any(L<=0.): return 1E100

    return -2.*np.log(L).sum()
```

⇐ only keep the events between 2.6 and 3.6

Now the binned histogram ⇐ is only used for plotting!

⇐ likelihood function: $L_i = f_s \cdot P_s + (1 - f_s) \cdot P_b$

⇐ likelihood value for each event

⇐ Protection for non-physical likelihood value

l306-example-04.py (partial)

UML FITTER EXAMPLE (II)

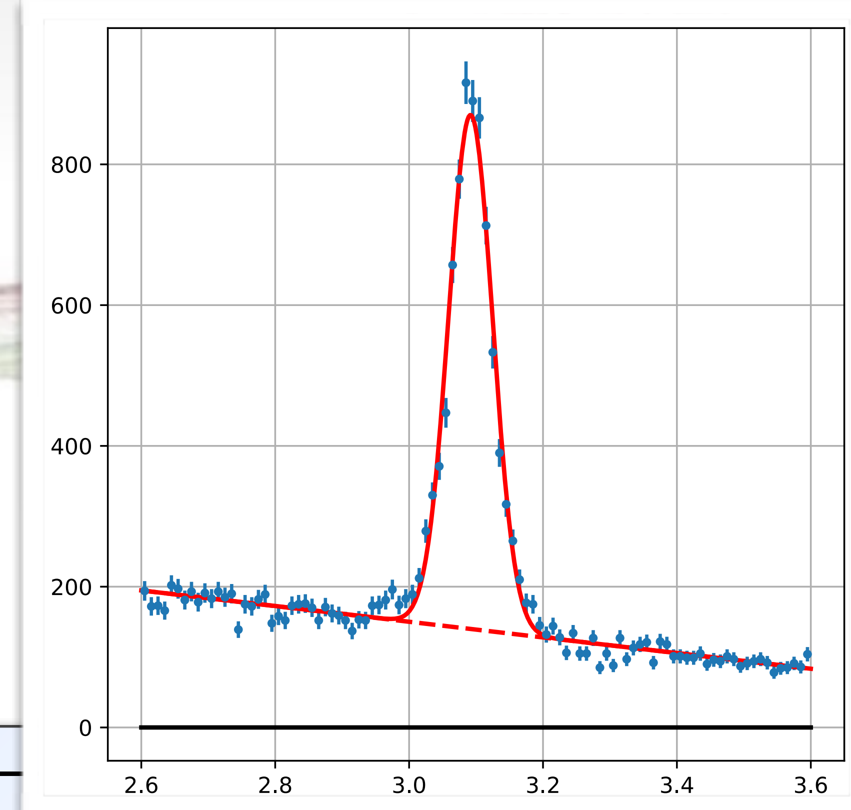
- The Minuit call is the same as before, only small modification to the plotting part.

```
m = Minuit(fcn, norm=6000., mean=3.09, sigma=0.04, c1=0.)
m.migrad()
m.minos()
m.print_param()

fig = plt.figure(figsize=(6,6), dpi=80)

plt.plot([xmin,xmax],[0.,0.],c='black',lw=2)
plt.errorbar(vx, vy, yerr = vyerr, fmt = '.')
cx = np.linspace(xmin,xmax,500)
cy = model(cx,m.values['fs'],m.values['mean'],m.values['sigma'],
              m.values['c1'])*xbinwidth*len(evt)
cy_bkg = model(cx,0.,m.values['mean'],m.values['sigma'],
         m.values['c1'])*xbinwidth*(len(evt)-m.values['norm'])
plt.plot(cx, cy, c='red',lw=2)
plt.plot(cx, cy_bkg, c='red',lw=2,ls='--')

. . . . . .
```

The likelihood function is in normalized to one, we have to times the # of total events & bin width! ⇓

l306-example-04.py (partial)

UML FITTER EXAMPLE (III)



- Just execute the code!

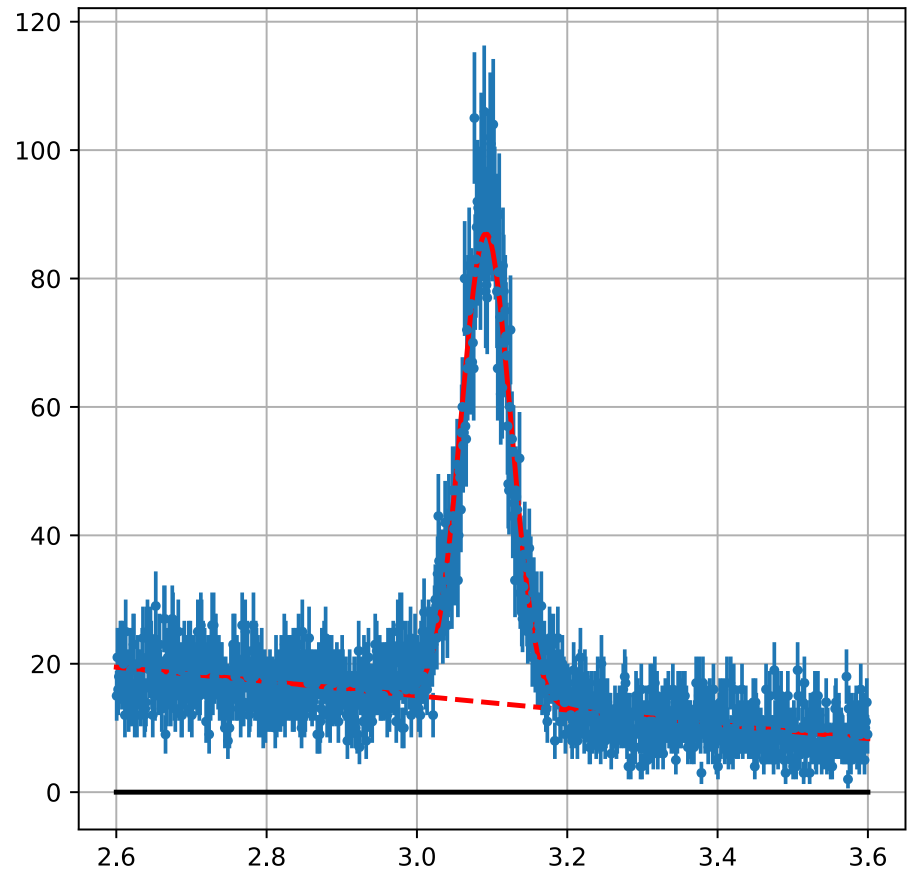
	Name	Value	Para Err	Err-	Err+
0	norm =	6022	88.38	-88.25	88.53
1	mean =	3.092	0.0005803	-0.0005807	0.0005799
2	sigma =	0.03290	0.0005753	-0.0005703	0.0005805
3	c1 =	-0.2299	0.002315	-0.002253	0.002381

- The results are consistent with the previous binned χ^2 fit:

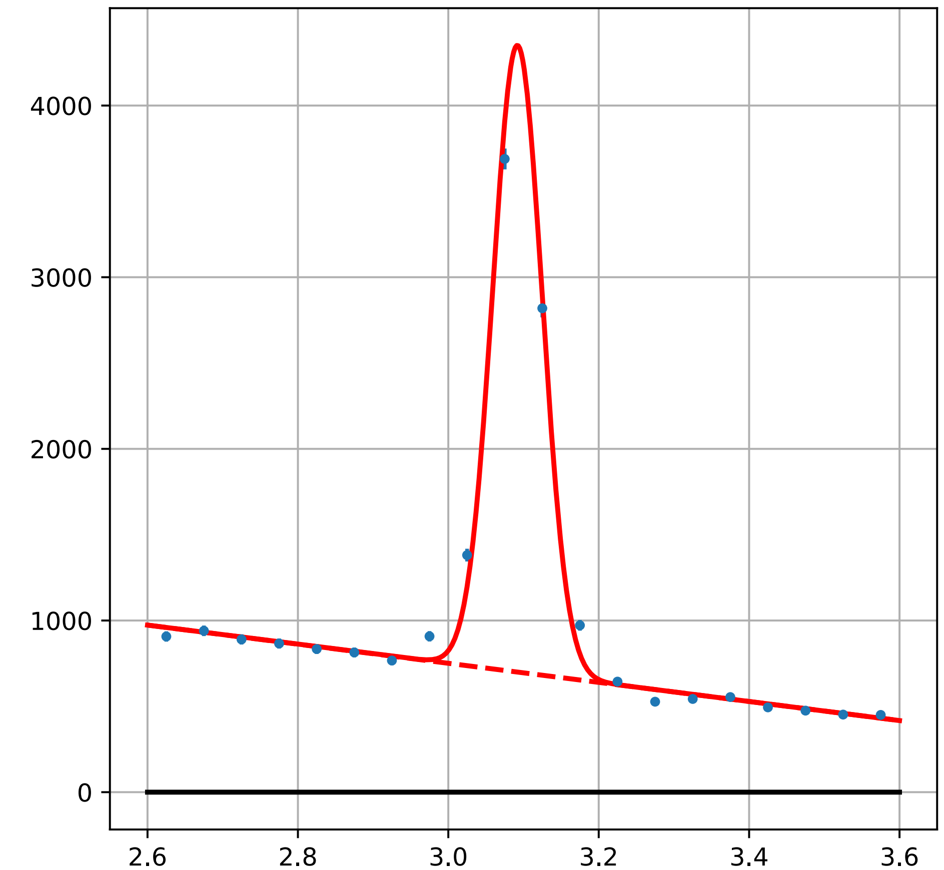
0	norm =	5984	96.34	-96.43	96.26
1	mean =	3.092	0.0005636	-0.0005661	0.0005615
2	sigma =	0.03282	0.0006026	-0.0005941	0.0006114
3	c0 =	193.1	2.668	-2.671	2.666
4	c1 =	-110.8	4.007	-4.002	4.013

REVISIT THE "PROBLEMS" — BINNING

- Now the fit does **NOT** depend on the binning anymore; the binned histogram is just for **making plots**.



1000 bins



20 bins

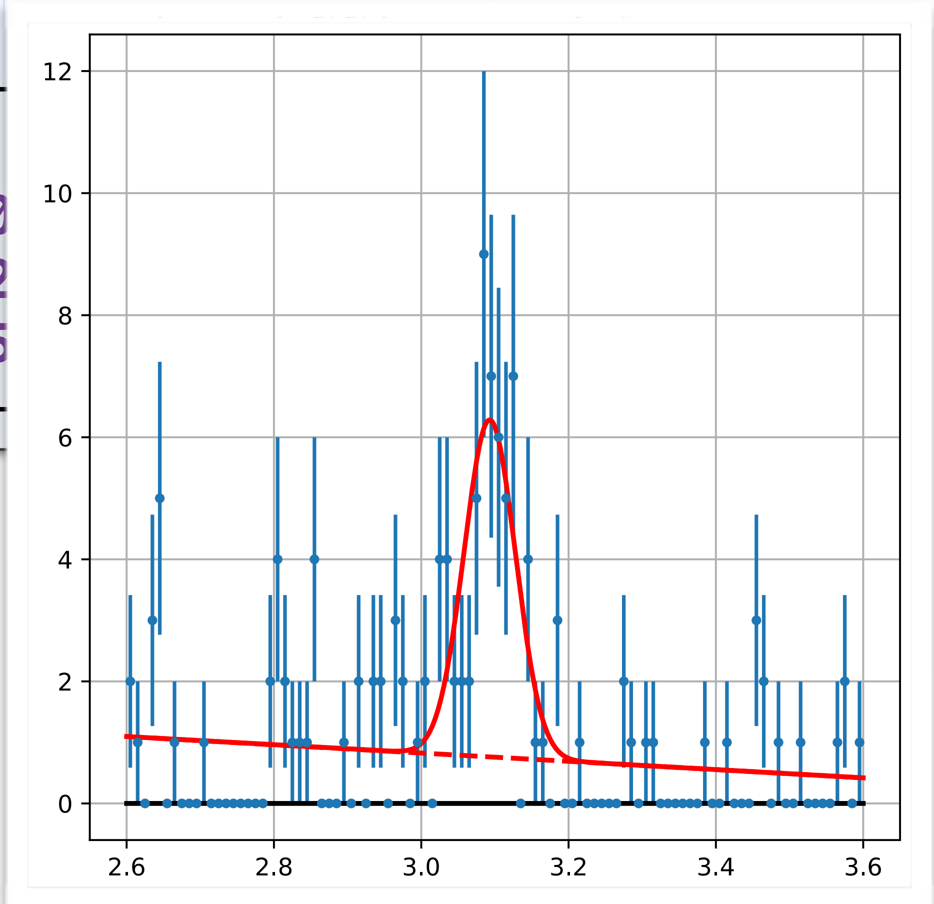
REVISIT THE "PROBLEMS" — NULL BINS

- Let's re-do the fit with only first **200** events again?

```
evt = np.load('dimuon.npy')[:200]
evt = evt[abs(evt-3.1)<0.5]
```

l306-example-04a.py (partial)

```
------------------------------------------------------------------
|       | Name  |  Value    | Para Err    |    Err-
------------------------------------------------------------------
|     0 |  norm =  48.21    |  7.292      | -7.250
|     1 |  mean =  3.093    | 0.00678     | -0.0070
|     2 | sigma =  0.03484  | 0.006607    | -0.0062
|     4 |    c1 = -0.2370   | 0.02504     | -0.0185
------------------------------------------------------------------
```



The background level is correctly estimated now.

COMMENT: MAXIMUM LIKELIHOOD ESTIMATOR

- The maximum likelihood estimator has "very good" statistical properties: it's **consistent**, **efficient**, and **robust**.
- ML estimators may have some bias, but they should decreases as N increases, if the selected PDF model is the correct one!
- The **efficiency of ML estimator is asymptotically 1**, when the size of observations approaches infinite: $N \rightarrow \infty$. ie. the variance of the ML estimator is very close to the ideal variance.
  - No other asymptotically unbiased estimator has asymptotic mean-squared error smaller than the ML estimator.
- **Nevertheless the ML is the widest used parameter estimator.**

→ The next question is what are the errors reported by the Minuit?

GAUSSIAN APPROXIMATION

- If we have a set of ***N*** independent measurements, whose PDFs are identical and are Gaussian, we have the model

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

- The likelihood function is

$$-2\ln L = \sum_{i=1}^{N}\frac{(X_i-\mu)^2}{\sigma^2} + N(\ln 2\pi + 2\ln\sigma)$$ *(just the x^2!)*

- The maximum likelihood estimate can be performed by an analytical minimization on μ (assuming σ is known):

$$\mu^{\text{est}} = \frac{1}{N}\sum_{i=1}^{N} X_i$$ *(Basically the sampling mean)*

- If σ^2 is also unknown, the ML estimate of σ^2 is:

$$(\sigma^{\text{est}})^2 = \frac{1}{N}\sum_{i=1}^{N}(X_i-\mu^{\text{est}})^2$$ *(mean-squares)*

ERROR ESTIMATION

- There are two approaches to determine parameter uncertainties.
- **Local error – the 2nd order partial derivatives with respect to the fit parameters around the minimum:**

$$C_{ij}^{-1} = -\frac{\partial^2 \ln L}{\partial\theta_i \partial\theta_j}$$

  - Under Gaussian approximation it equals to the covariance matrix;
  - May lead to underestimated errors with finite samples.
- **Evaluation of –2ln$L$ values around the maximum point of likelihood function.**
  - Leads to usual error matrix in a Gaussian model
  - May lead to asymmetric errors.

MIGRAD/HESSE command under minuit

MINOS command under minuit

ERROR ON MEAN?

- Let's practice the calculation with second derivatives!
- Remember the likelihood function with the assumption of Gaussian models:

$$-2\ln L = \sum_{i=1}^{N} \frac{(X_i - \mu)^2}{\sigma^2} + N(\ln 2\pi + 2\ln\sigma)$$

- The error on the mean ***μ*** can be estimated by

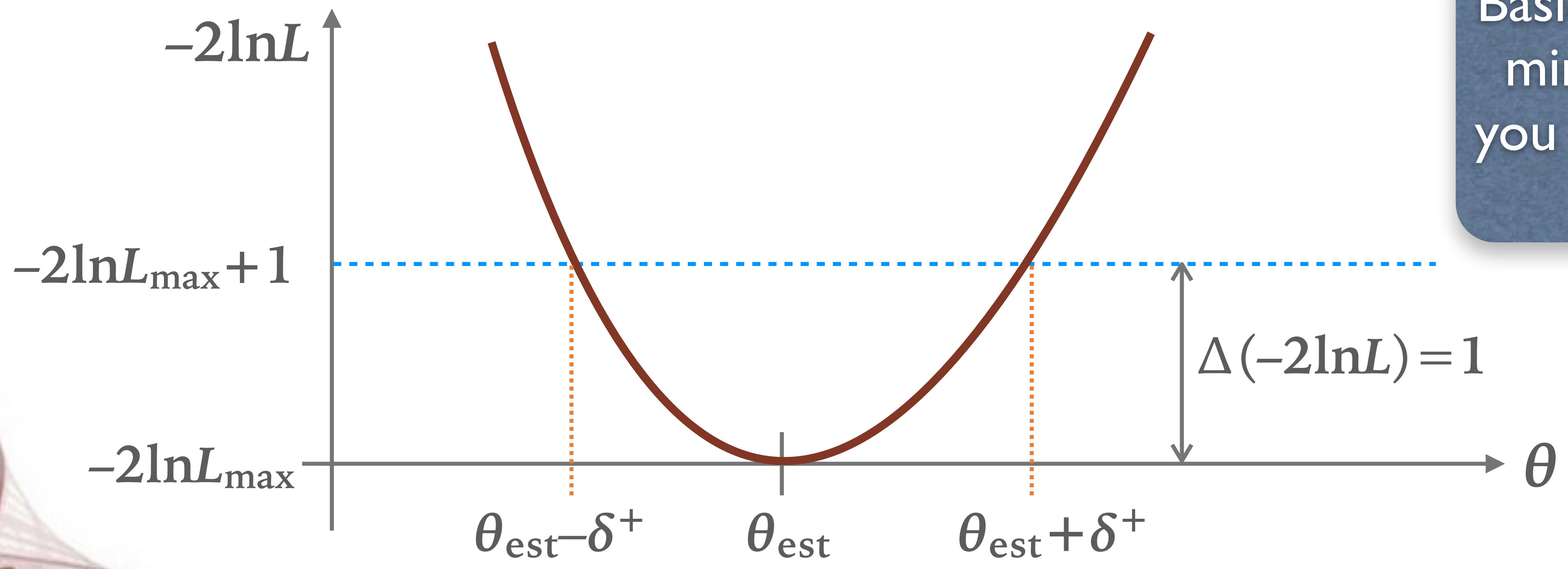
$$C_\mu^{-1} = \frac{1}{\delta_\mu^2} = -\frac{\partial^2 \ln L}{\partial \mu^2} = \frac{N}{\sigma^2}$$

- And it just gives us the usual estimation of **"error on mean"**:

$$\delta_\mu = \frac{\sigma}{\sqrt{N}}$$

ASYMMETRIC ERROR

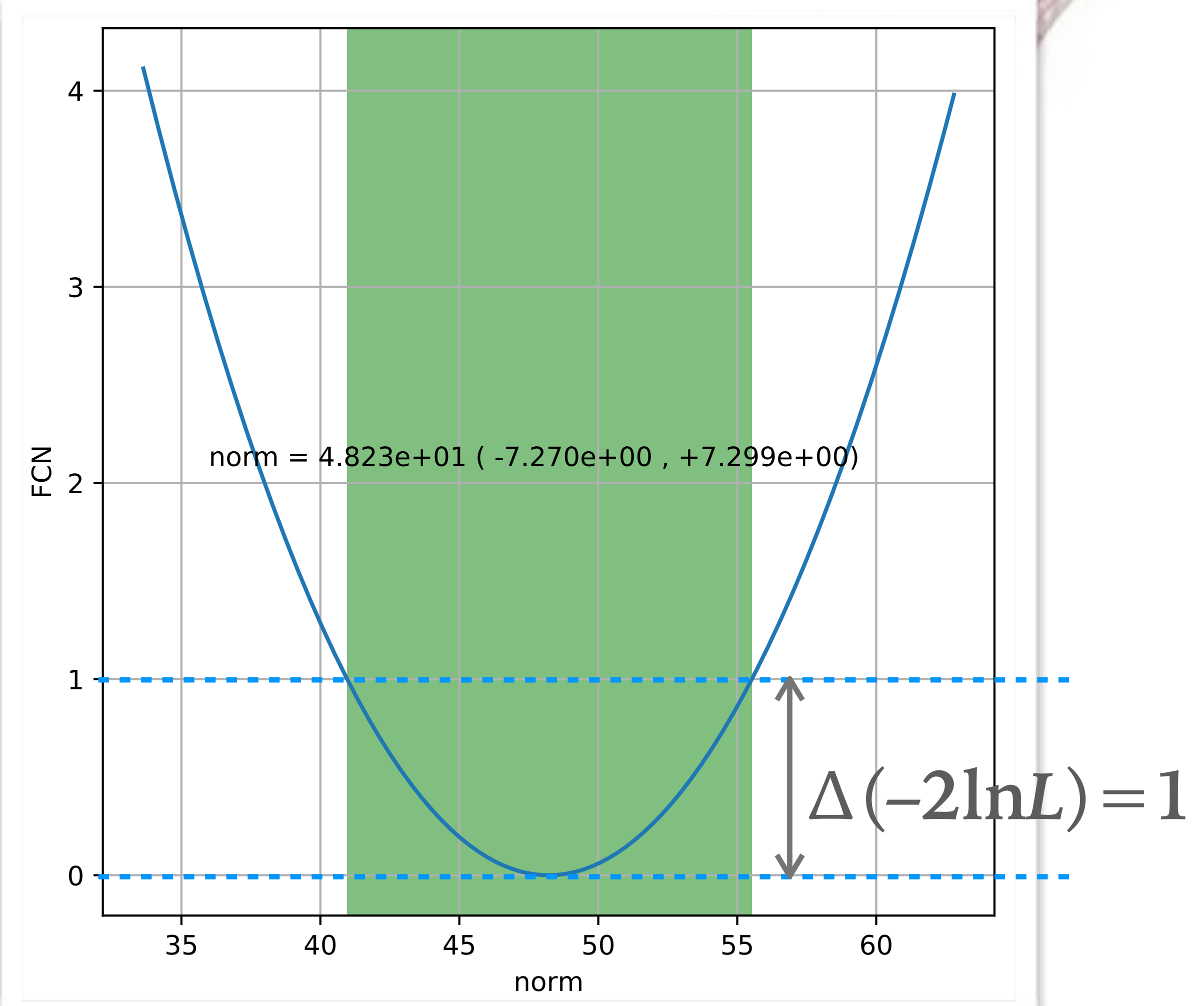
- If the **$-2\ln L$** function is close to a parabolic shape, the derivatives can be approximated by parameter excursion ranges.
- A **"n-σ"** error can be determined by the range around the Likelihood maximum for which the **$-2\ln L$** value increases by **n^2**:
  - The errors can be asymmetric for the positive and negative side!
  - It is identical to the **σ** of Gaussian PDF.



Basically this is what minuit does when you call the MINOS command.

SCAN OVER LIKELIHOOD FUNCTION

- The iminuit package has provide a simple tool to produce a scan over the FCN function you provided. e.g.
- Minos reported value is **norm = 48.21 +7.313/–7.250**, which is consistent with the result from a direct **profile likelihood scan**.




```
fig = plt.figure(figsize=(6,6), dpi=80)
m.draw_mnprofile('norm', bins=1000, subtract_min=True)
plt.show()
```

l306-example-04b.py (partial)

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 it as **fs $\rightarrow$ 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 again.

A MUCH CLEANER SAMPLE?

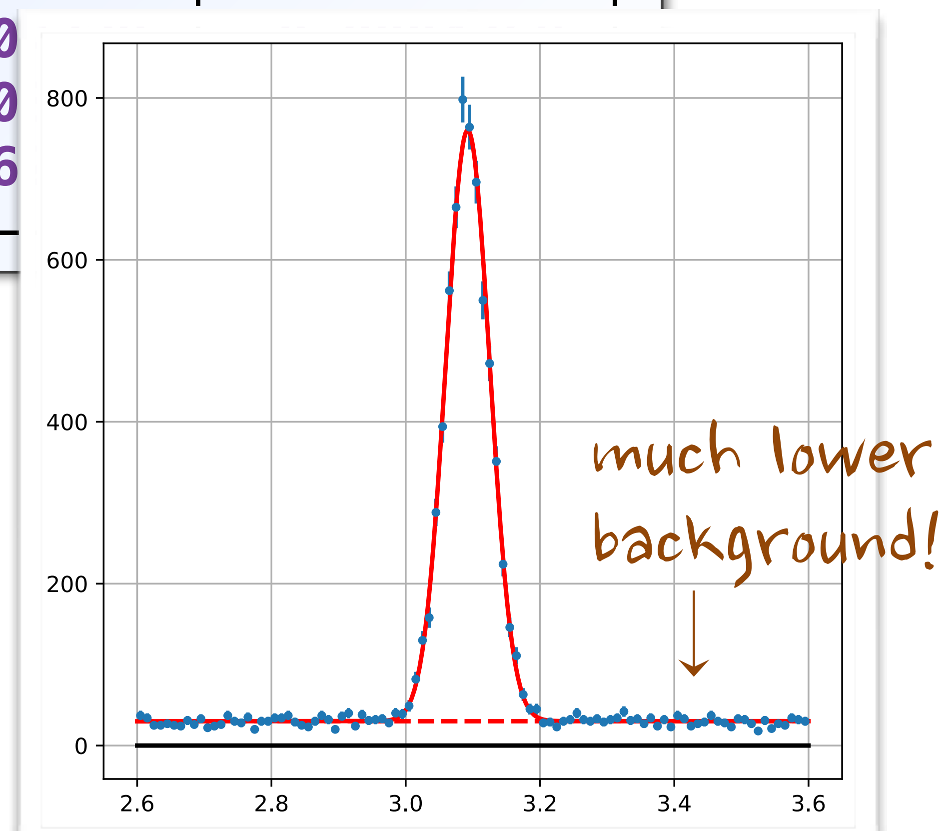
- Just use another example data set with S/N ~ 2: `clean_data.npy`

```python
evt = np.load('clean_data.npy')
```

l306-example-04c.py (partial)

```
------------------------------------------------------------------
|      | Name  |  Value     | Para Err   |   Err-      |   Err+     |
------------------------------------------------------------------
|    0 |  norm = 5998       |  52.61     | -52.82      |  52.41     |
|    1 |  mean = 3.092      |  0.0004748 | -0.000
|    2 | sigma = 0.03277    |  0.0003942 | -0.000
|    3 |    c1 = 0.007229   |  0.06743   | -0.056
-------------------------------------------------------
```

  - Look at the error of "norm", it's actually too small **~0.9%**. The uncertainty cannot be smaller then the Poisson error (square-root of the "norm", which is **~1.3%**).



THE EXTENDED MAXIMUM LIKELIHOOD ESTIMATOR

- The likelihood function for each event should be modified to

$$L_i = n_s \cdot P_s(x_i; \mu, \sigma) + n_b \cdot P_b(x_i; c_1)$$

The best solution by maximizing the total likelihood:

Total observed event N is fixed:

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

Or by minimizing the value of

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

A constant, can be thrown away.

P_s (P_b) : signal (background) PDF

n_s (n_b) : signal (background) yields

μ, σ, c_1 : fitting parameters to be resolved by the estimator

EXTENDED UML FITTER

```
evt = np.load('dimuon.npy')
evt = evt[abs(evt-3.1)<0.5]

xmin, xmax, xbinwidth = 2.6, 3.6, 0.01
vy,edges = np.histogram(evt, bins=100, range=(xmin,xmax))
vx = 0.5*(edges[1:]+edges[:-1])
vyerr = vy**0.5

def model(x, ns, nb, mean, sigma, c1):

    linear = (1. + c1*x)/((xmax-xmin) + c1*(xmax**2-xmin**2)/2.)

    gaussian = 1./(2.*np.pi)**0.5/sigma * \
        np.exp(-0.5*((x-mean)/sigma)**2)

    return ns*gaussian + nb*linear

def fcn(ns, nb, mean, sigma, c1):

    L = model(evt, ns, nb, mean, sigma, c1)
    if np.any(L<=0.): return 1E100

    return 2.*(ns+nb)-2.*np.log(L).sum()
```

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

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

l306-example-05.py (partial)

EXTENDED UML FITTER (II)

- Also need to modify the minuit call and plotting code a little bit.

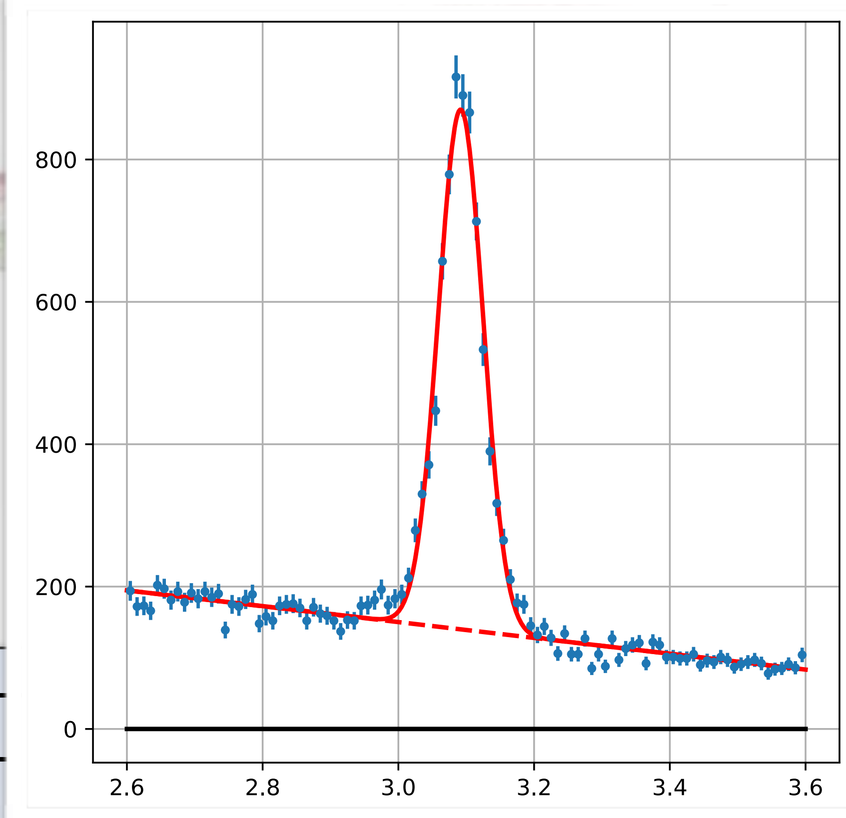
```
m = Minuit(fcn, ns=6000., nb=14000., mean=3.09, sigma=0.04, c1=0.)
m.migrad()
m.minos()
m.print_param()

fig = plt.figure(figsize=(6,6), dpi=80)

plt.plot([xmin,xmax],[0.,0.],c='black',lw=2)
plt.errorbar(vx, vy, yerr = vyerr, fmt = '.')
cx = np.linspace(xmin,xmax,500)
cy = model(cx,m.values['ns'],m.values['nb'],m.values['mean'],
              m.values['sigma'],m.values['c1'])*xbinwidth
cy_bkg = model(cx,0.,m.values['nb'],m.values['mean'],
                     m.values['sigma'],m.values['c1'])*xbinwidth
plt.plot(cx, cy, c='red',lw=2)
plt.plot(cx, cy_bkg, c='red',lw=2,ls='--')

plt.grid()
plt.show()
```

⇑ initial ns and nb

The likelihood function is ⇓ in normalized to Ns+Nb now!

l306-example-05.py (partial)

EXTENDED UML FITTER (III)



- Let's just try it:

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

	Name	Value	Para Err	Err-	
0	ns =	6022	98.04	-97.89	98.4
1	nb =	1.39E+04	132.2	-132	132.7
2	mean =	3.092	0.0005802	-0.0005801	0.0005806
3	sigma =	0.03290	0.0005746	-0.0005702	0.0005806
4	c1 =	-0.2299	0.002315	-0.002252	0.002382

- The results are (*almost*) the same as the standard UML fit:

0	norm =	6022	88.38	-88.25	88.53
1	mean =	3.092	0.0005803	-0.0005807	0.0005799
2	sigma =	0.03290	0.0005753	-0.0005703	0.0005805
3	c1 =	-0.2299	0.002315	-0.002253	0.002381

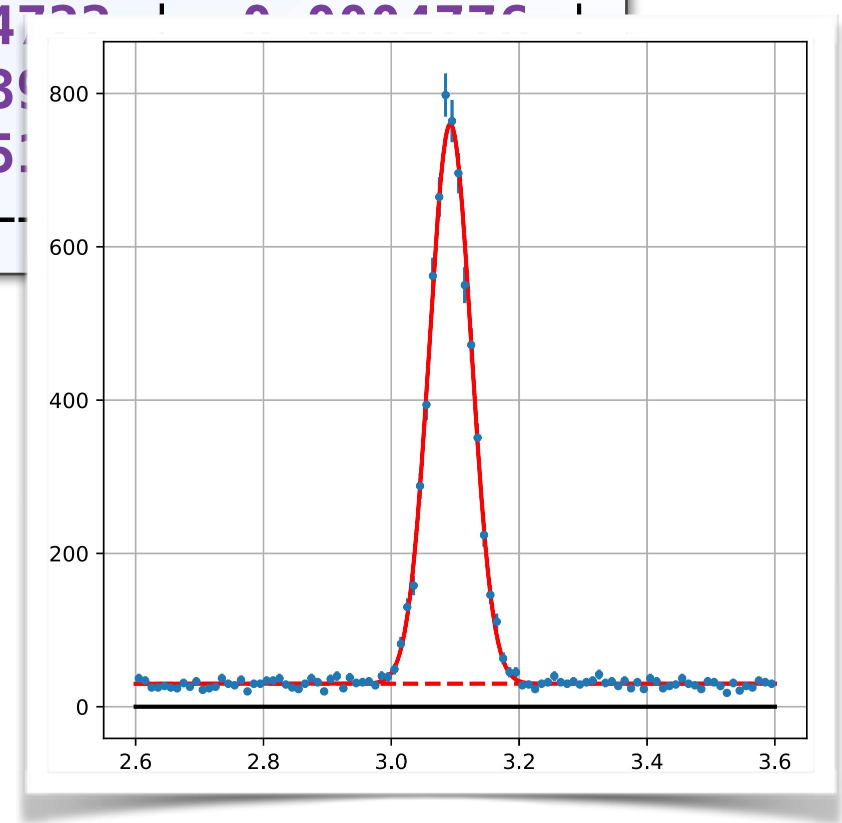
EXTENDED UML FITTER (IV)

- Try the clean sample again:

```
evt = np.load('clean_data.npy')
```

l306-example-05a.py (partial)

	Name	Value	Para Err	Err−	Err+
0	ns = 5998		82.24	−82.25	82.26
1	nb = 3003		61.39	−52.82	52.41
2	mean = 3.092		0.0004748	[illegible]	[illegible]
3	sigma = 0.03277		0.0003941	[illegible]	[illegible]
4	c1 = 0.007462		0.06743	[illegible]	[illegible]



– Since this extended ML fit includes the Poisson error to the total # of events, the uncertainties can be correctly estimated (>*square root of the event counts!*).

HOW DO YOU KNOW THE ERRORS ARE CORRECT?

- Although we have commenting that the reported errors of the yields cannot be smaller than the Poisson variance, since such a particle detecting is a Poisson process, but we have not yet fully prove the errors given by Minuit does match to the standard **"one-sigma"** error.
- Here we just want to introduce a typical method to verify this. This is what we usually call the **pseudo experiments**.
- This means, we can generate many sets of "pseudo (toy) data" using the random numbers, while these data should follow exactly the expected statistical distributions. Then we use our estimator to fit these toy data sets and obtain their associated estimates and uncertainties.

HOW DO YOU KNOW THE ERRORS ARE CORRECT? (II)

- If our tool is reporting the correct mean and errors, the resulting estimate from each fit should agree with the input values up to the fluctuations allowed by the reported errors.
- One of the typical way to verify this is to calculate the **"pulls"**, which are defined by

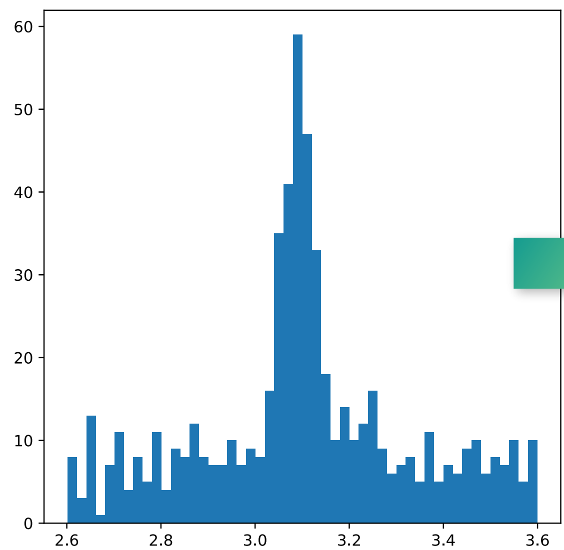
$$P_i = \left(\frac{\mu_i^{\text{est}} - \mu^0}{\sigma_i^{\text{est}}} \right)$$

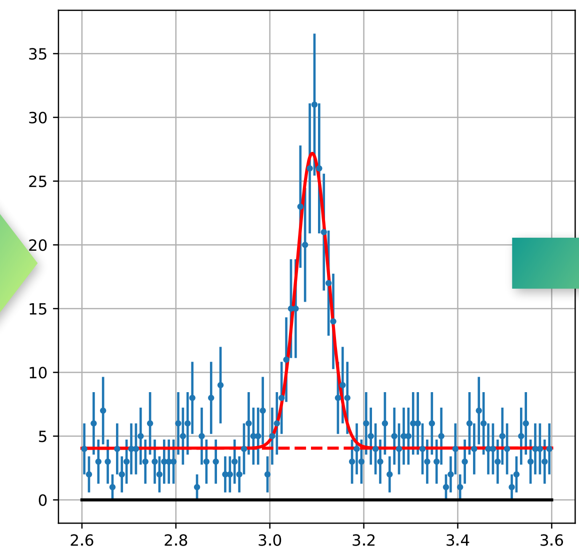
Indices i means ith set of pseudo data

- If everything is correctly implemented (including both the pseudo data generation & estimator), the pull ***P*** should just distribute like a **standard normal distribution** with mean zero and width one.
- Let's practice this method with our extended ML estimator!

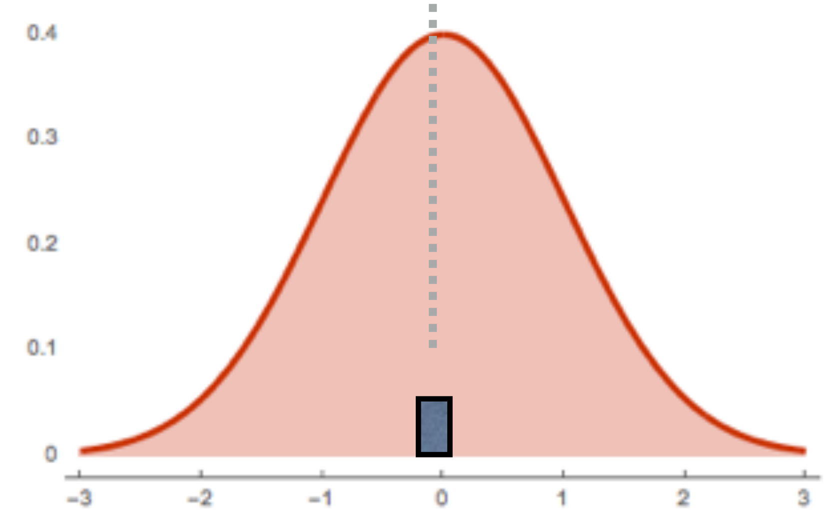
GENERATING & FITTING

- Here we introduce a very simple test model and perform the study just mentioned in the previous slides.
- For each **pseudo experiment**:





$$P_i = \left(\frac{\mu_i^{\text{est}} - \mu^0}{\sigma_i^{\text{est}}} \right)$$



Repeating the generation and fits, see if the resulting distribution agrees with a standard Gaussian or not!

PSEUDO EXPERIMENT

- Here are a simple example code to perform a pseudo experiment: generating events and perform a fit afterwards.
- The interface with Minuit is the same as the previous example.

```
S = np.random.randn(200)*0.03290+3.092
B = np.random.rand(400)+2.6
evt = np.hstack([S,B])
np.random.shuffle(evt)

xmin, xmax, xbinwidth = 2.6, 3.6, 0.01
vy,edges = np.histogram(evt, bins=100, range=(xmin,xmax))
vx = 0.5*(edges[1:]+edges[:-1])
vyerr = vy**0.5

def model(x, ns, nb, mean, sigma, c1):
    . . . . . .

def fcn(ns, nb, mean, sigma, c1):
    . . . . . .
```

⇐ generation of "toy" events

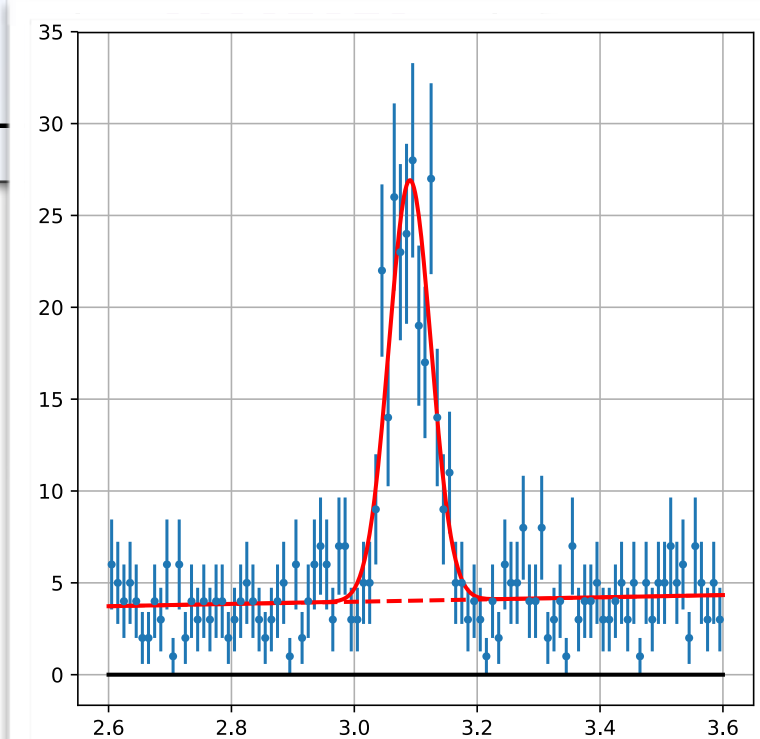
l306-example-06.py (partial)

PSEUDO EXPERIMENT (II)

- The rest of the code is the same as the previous example, here we just run it and show you the results:

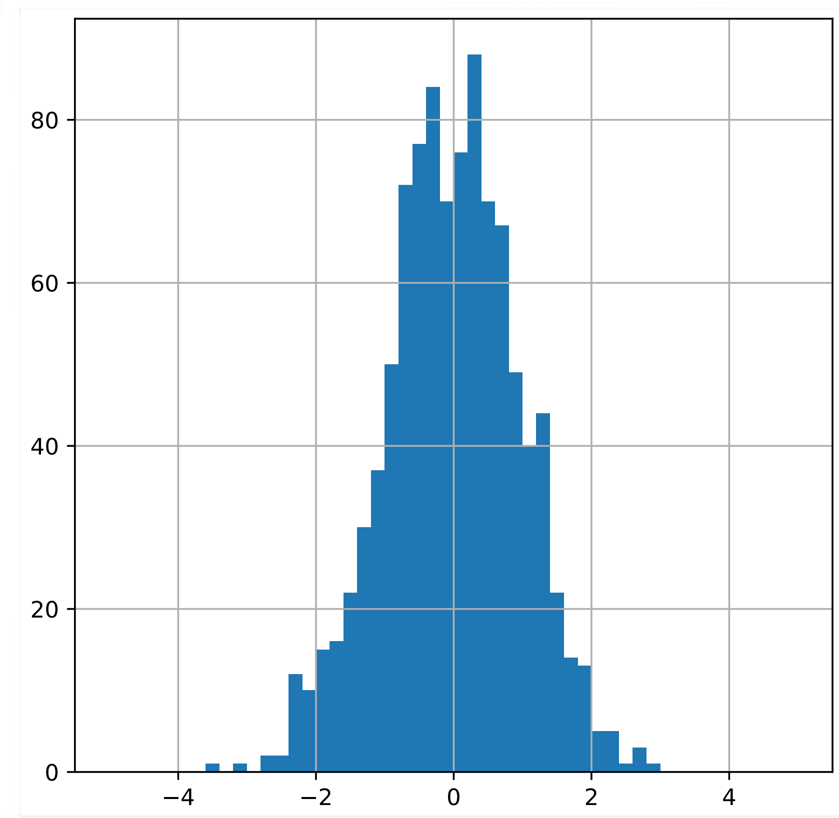
	Name	Value	Para Err	Err-	Err+
0	ns =	196.9	16.88	-16.47	17.33
1	nb =	403.1	22.17	-21.87	22.47
2	mean =	3.09	0.003184	-0.003183	0.003197
3	sigma =	0.03431	0.002655	-0.002518	0.002816
4	c1 =	0.2828	0.599	-0.599	[illegible]

Here are the result of ONE pseudo experiment, now the next to verify the fluctuation of # of signal and its error.



PSEUDO EXPERIMENT (III)

- We need to repeat generation + fit for many iteration, and verify the resulting distribution.




```
values = np.zeros(1000)
errors = np.zeros(1000)

for idx in range(len(values)):
    S = np.random.randn(np.random.poisson(200.))*0.033+3.092
    B = np.random.rand(np.random.poisson(400.))+2.6
    evt = np.hstack([S,B])

    m = Minuit(fcn, ns=200., nb=400.,
        mean=3.092, sigma=0.033, c1=0.)
    m.migrad()

    values[idx] = m.values['ns']
    errors[idx] = m.errors['ns']

pull = ((values-200.)/errors)[errors!=0.]

print('Pull mean:',pull.mean())
print('Pull width:',pull.std())
. . . . . .
```

⇐ generation

⇐ fit

⇓ Nearly a standard Gaussian!

```
Pull mean: -0.00900125913
Pull width: 1.03899390174
```

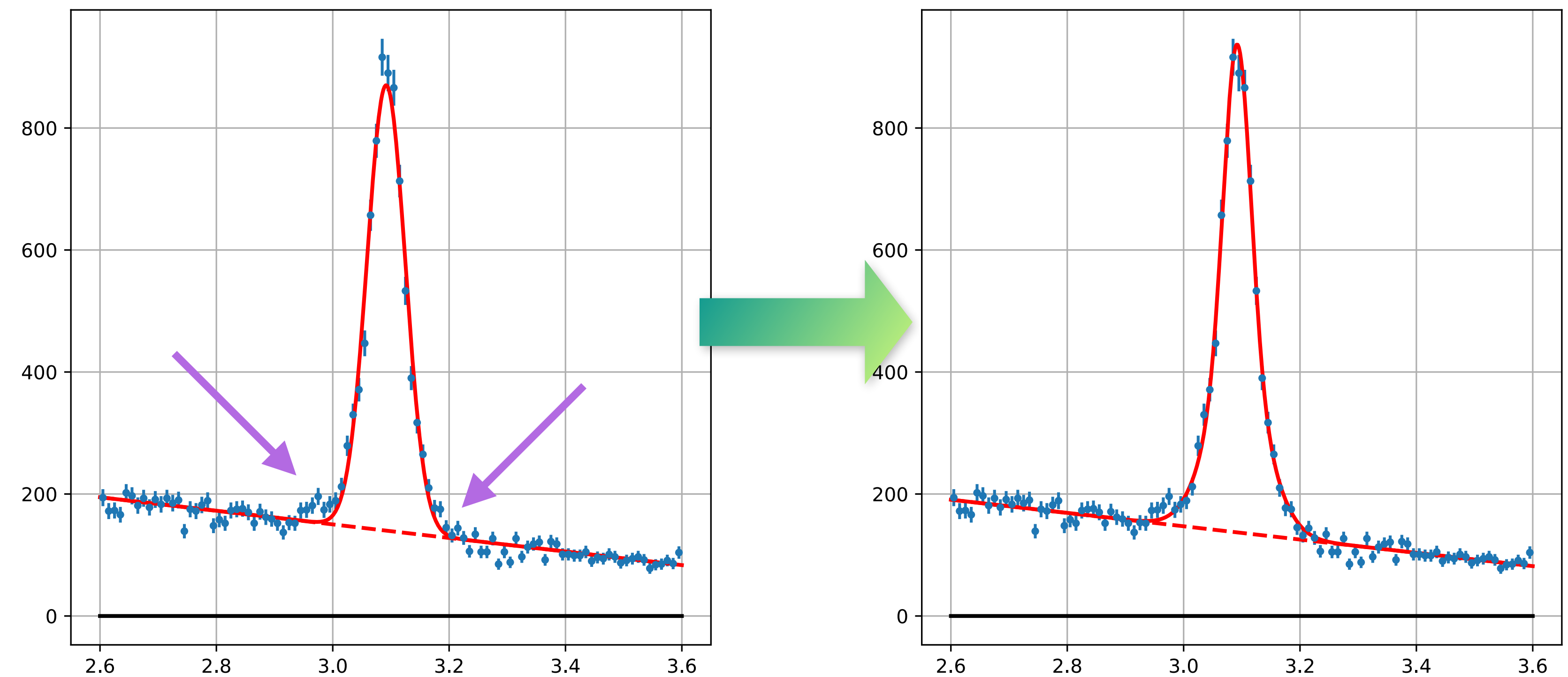
l306-example-06a.py (partial)

FINAL COMMENT

- The **(extended) unbinned maximum likelihood** method is the best estimator for most of the parameter extraction problems.
- It should provide a proper estimate of the parameters and well as the associated statistical uncertainties, as far as your *model is correct*!
- Although we have present this with a typical analysis of "bump-like" data and use it to extract the hidden parameter of the peak, but the method can be adopted to many other different applications.
- There are still many related topics can be discussed (e.g. confidence level estimation, upper/lower limit, hypothesis tests), but we will stop here and keep them for your own future study.

HANDS-ON SESSION

- **Practice 01:**
  Maybe you "feel" the model to the J/ψ mass peak is not good enough? There might be some tails near the peak and it cannot be described by a single Gaussian.
- Please extend the model by adding the second Gaussian to the signal peak and see if you can get the resulting plot as below?



HANDS-ON SESSION

- **Practice 02:**
  Go back to the original mass plot with a wider range. I have claimed there is in fact a second peak of the "ψ(2S)" particle at 3.69 GeV. It is an excited state of the big J/ψ peak.
- Try to perform a fit to it instead of the big peak!