23 Jan, 2024
Tirana

Data Science Applications in Physics, Balkan School in Tirana 2024

The replica method
application using the ROOT framework

Albi Kerbizi
University of Trieste and INFN Trieste Section

POLFRAG

Outlook

Introduction

recall of the concept of confidence interval

The replica method

generation of pseudodata,
study of uncertainties and correlations,
building uncertainty bands

Implementation in ROOT

basics of ROOT,
hands-on session (using a proposed program)

The concept of confidence interval

- Suppose we are interested in estimating the physical quantity θ from the measurement of $x_1, x_2, .., x_n$ indipendent random variables

→ introduce an estimator $t(x_1, x_2, .., x_n)$ such that $E[t] = \theta_0$
θ_0 the true value of the physical quantity θ

→ thus evaluating t on the measured sample $(x_1, x_2, .., x_n)$ gives an estimate of θ
→ a second measurement $(x_1', x_2', .., x_n{}')$ would however give another estimate of θ

The concept of confidence interval

- Suppose we are interested in estimating the physical quantity θ from the measurement of $x_1, x_2, .., x_n$ indipendent random variables

→ introduce an estimator $t(x_1, x_2, .., x_n)$ such that $E[t] = \theta_0$
θ_0 the true value of the physical quantity θ

→ thus evaluating t on the measured sample $(x_1, x_2, .., x_n)$ gives an estimate of θ
→ a second measurement $(x'_1, x'_2, .., x_n{}')$ would however give another estimate of θ

- To quantify these fluctuations → Confidence Interval (CI) found as follows

  i. chose a Confidence Level (CL) γ ($\gamma = 0.68$ or $\gamma = 0.90$ or $\gamma = 0.95$)
  ii. Find t_a and t_b such that $P(t_a < t < t_b) \equiv \int_{t_a}^{t_b} dt\, f(t) = \gamma$
  iii. Since it is $P(t_a < t < t_b) = P(\theta_a < \theta < \theta_b)$, find the CI $[\theta_a, \theta_b]$ such that
  $P(\theta_a < \theta < \theta_b) = \gamma$

  ex: Confidence Interval with CL 68% ($\gamma = 0.68$) → interval such that the probability to find there the physical quantity is 68%

The concept of confidence interval

example

- Example: we measure a sample $x_1, x_2, .., x_n$ of n independent random variables each distributed according to the gaussian distribution $N(\mu, \sigma)$
  mean value $\mu \rightarrow$ unknown physical quantity
  standard deviation $\sigma \rightarrow$ known

Introduce the estimator

$t(x_1, x_2, .., x_n) = \frac{1}{n}\sum_{i=1}^{n} x_i \equiv \bar{x} \rightarrow$ gives an estimate of μ

The concept of confidence interval

example

- Example: we measure a sample $x_1, x_2, .., x_n$ of n independent random variables each distributed according to the gaussian distribution $N(\mu, \sigma)$
  mean value $\mu \rightarrow$ unknown physical quantity
  standard deviation $\sigma \rightarrow$ known

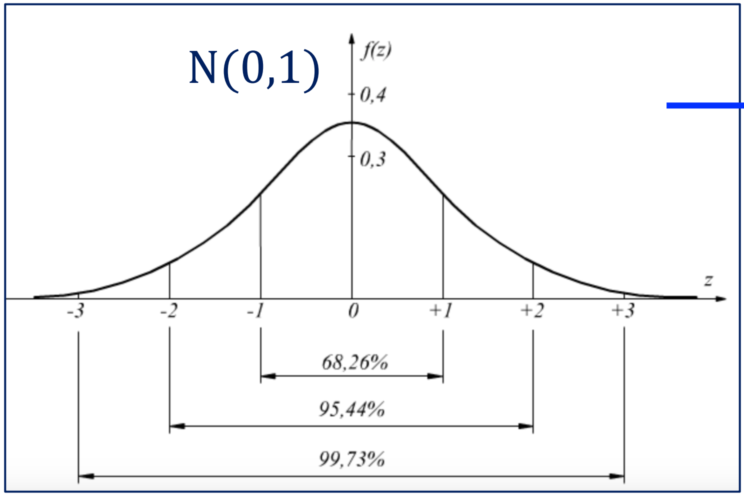
Introduce the estimator

$t(x_1, x_2, .., x_n) = \frac{1}{n}\sum_{i=1}^{n} x_i \equiv \bar{x} \rightarrow$ gives an estimate of μ

To find the CI at 68% CL

we know that the distribution of $\bar{x}$ is $N(\mu, \sigma/\sqrt{n})$

thus introduce $z = \frac{\bar{x}-\mu}{\sigma/\sqrt{n}} \rightarrow$ distributed as $N(0,1)$



$$P(-1 < z < 1) \simeq 0.68$$

$$\rightarrow P\left(-1 < \frac{\bar{x}-\mu}{\frac{\sigma}{\sqrt{n}}} < 1\right) \simeq 0.68$$

The CI at 68%CL is

$$[\bar{x} - \frac{\sigma}{\sqrt{n}}, \quad \bar{x} + \frac{\sigma}{\sqrt{n}}]$$

The concept of confidence interval

example

- Example: we measure a sample $x_1, x_2, .., x_n$ of n independent random variables each distributed according to the gaussian distribution $N(\mu, \sigma)$
  mean value $\mu \rightarrow$ unknown physical quantity
  standard deviation $\sigma \rightarrow$ known

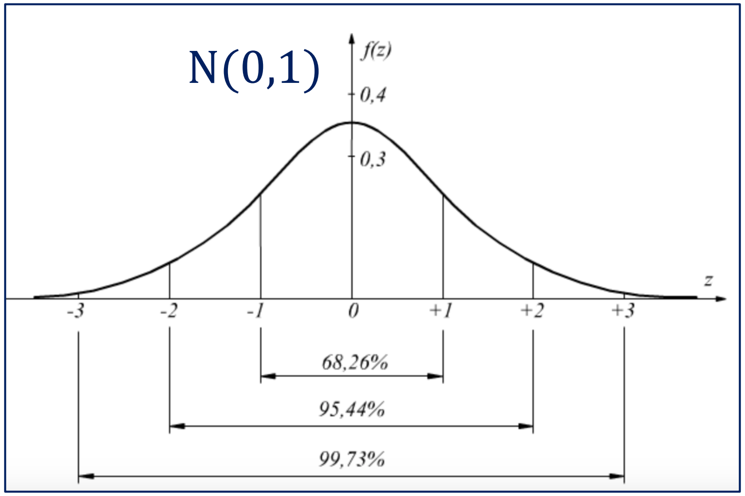
Introduce the estimator

$t(x_1, x_2, .., x_n) = \frac{1}{n}\sum_{i=1}^{n} x_i \equiv \bar{x} \rightarrow$ gives an estimate of μ

To find the CI at 68% CL

we know that the distribution of $\bar{x}$ is $N(\mu, \sigma/\sqrt{n})$

thus introduce $z = \frac{\bar{x}-\mu}{\sigma/\sqrt{n}} \rightarrow$ distributed as $N(0,1)$

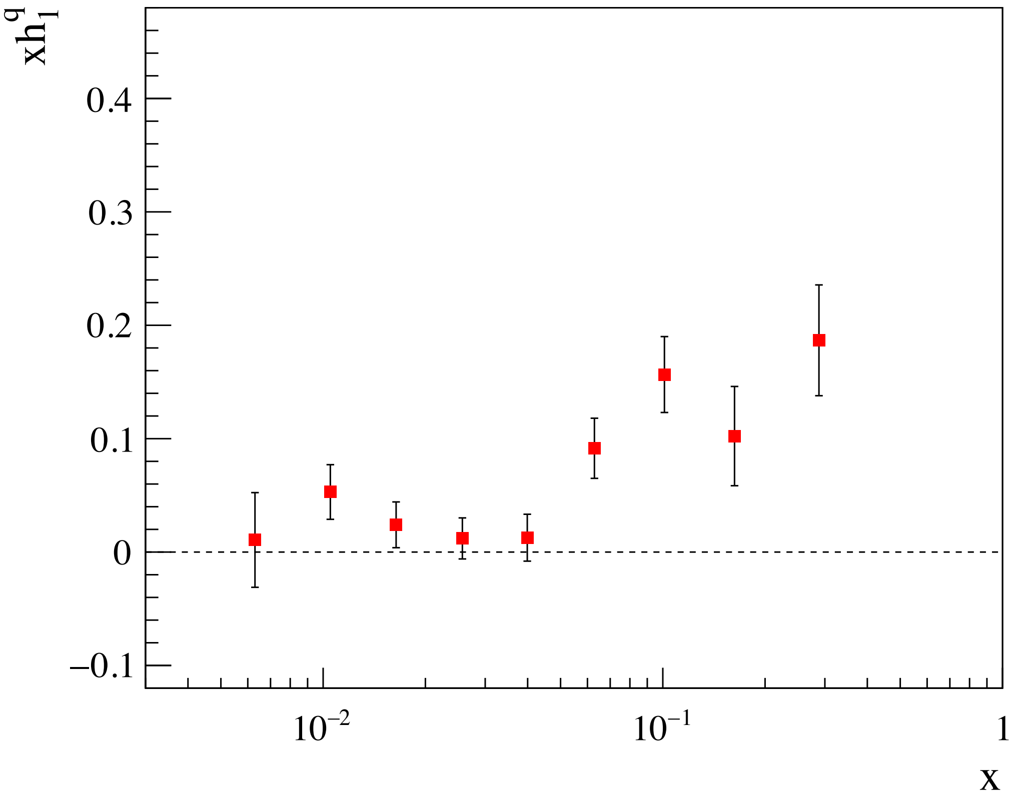


Condition:
we must know the distribution $f(t)$ of the estimator t!

In general $f(t)$ not known $\rightarrow$ must rely e.g. on MC methods, like the replica method
-> this presentation!
not for one point but for many -> «confidence region» or «confidence band»

Fit of data

Now we consider a more complicated exercise
the fit of a data sample!



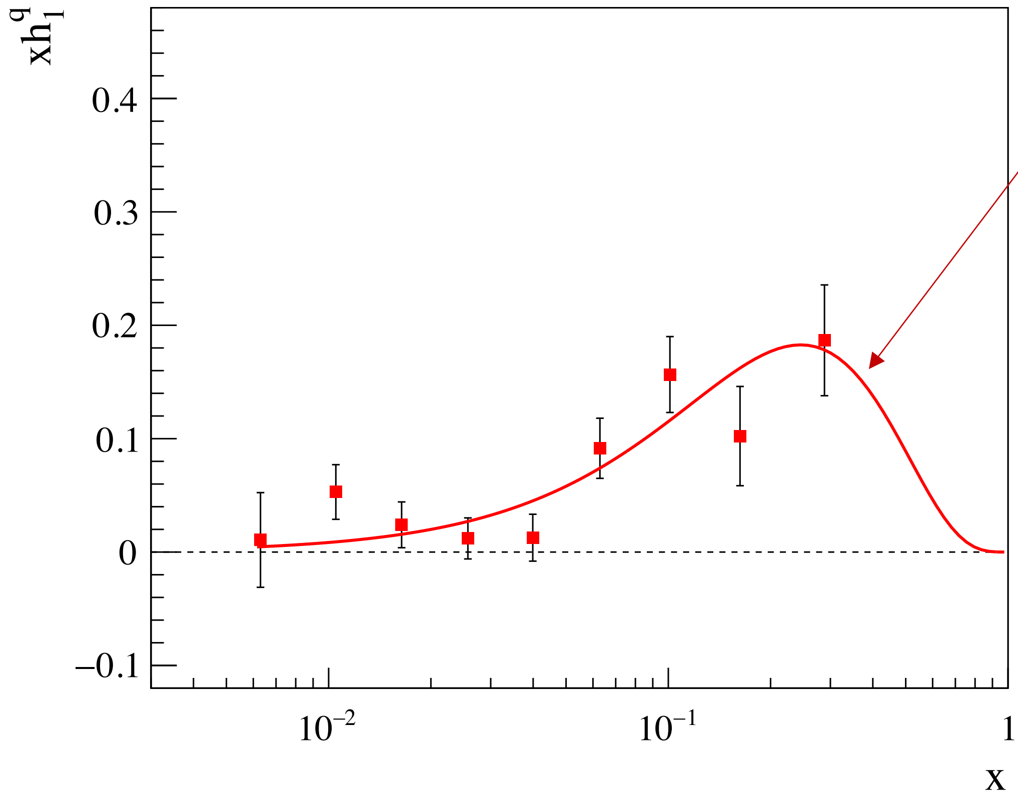
Fit of data

Now we consider a more complicated exercise
the fit of a data sample!

To fit the data points we use the function

$$xh_1^q(x) = a\,x^b\,(1-x)^4 \equiv y(x)$$

a, b free parameters



Fitted function using ROOT (X^2 minimation)

Parameters

$$\hat{a} \pm \sigma_{\hat{a}} = 3.47 \pm 1.52$$
$$\hat{b} \pm \sigma_{\hat{b}} = 1.31 \pm 0.19$$
$$X^2/\text{ndf} = 10.4/7$$

Notation: the values of parameters from fit result are indicated with a « $\hat{}$ »

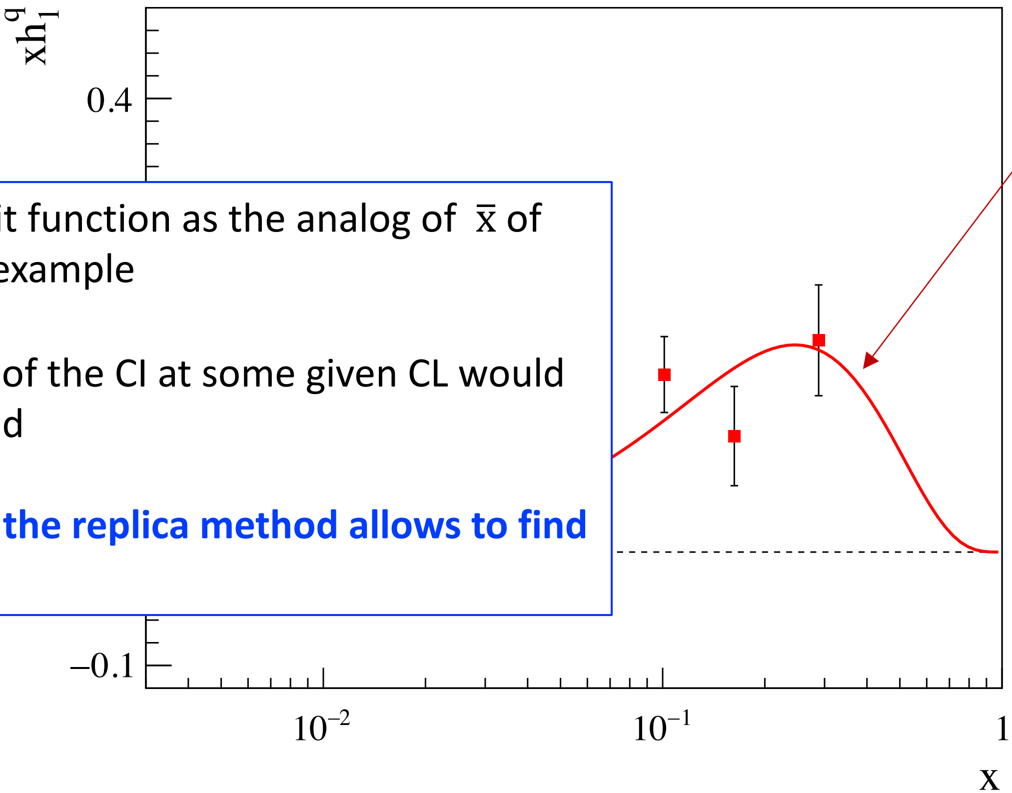
Fit of data

Now we consider a more complicated exercise
the fit of a data sample!

To fit the data points we use the function

$$xh_1^q(x) = a\,x^b\,(1-x)^4 \equiv y(x)$$

a, b free parameters



Think of the fit function as the analog of $\bar{x}$ of the previous example

→ the analog of the CI at some given CL would then be a band

let's see how the replica method allows to find the band!

Fitted function using ROOT (X^2 minimation)

Parameters

$$\hat{a} \pm \sigma_{\hat{a}} = 3.47 \pm 1.52$$
$$\hat{b} \pm \sigma_{\hat{b}} = 1.31 \pm 0.19$$
$$X^2/\text{ndf} = 10.4/7$$

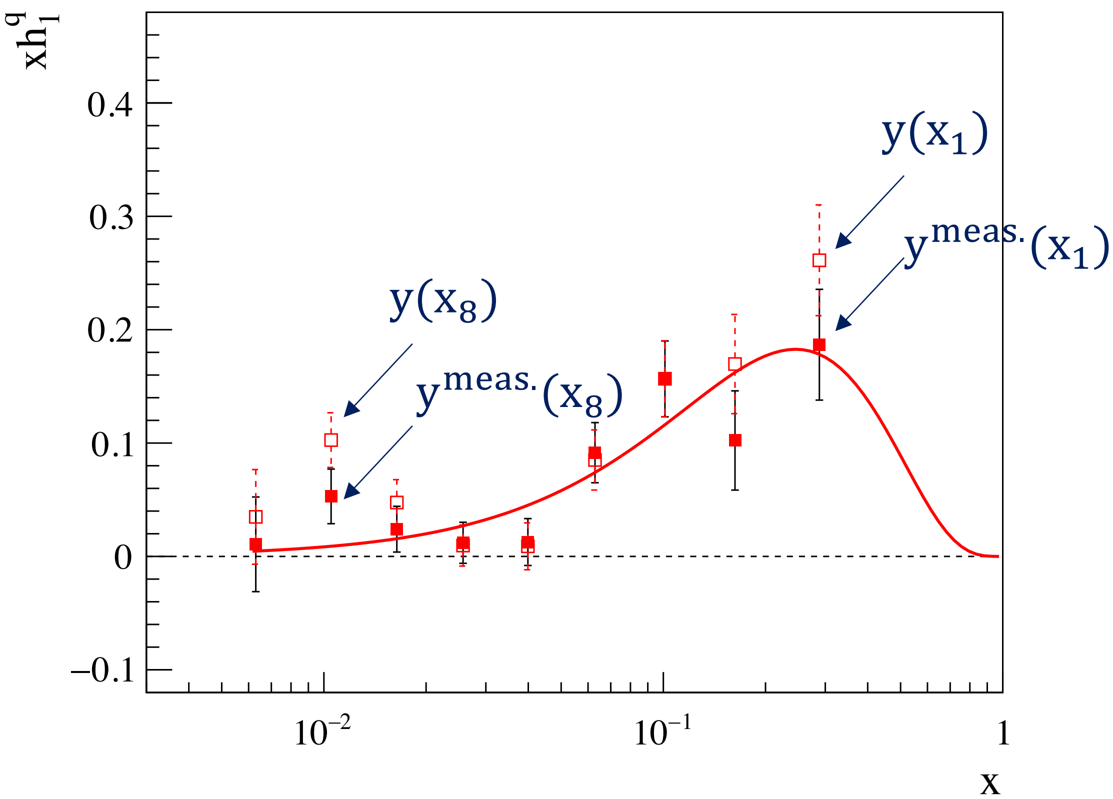
Notation: the values of parameters from fit result are indicated with a « $\hat{}$ »

Constructing the replica

i. Consider a data point $y_i^{meas.}$ with statistical uncertainty σ_i
ii. Generate a new point y_i («pseudo-data») according to the gaussian distribution

$$N(y_i; y_i^{meas.}, \sigma_i) = \exp\left[-\frac{(y_i - y_i^{meas.})^2}{2\sigma_i^2}\right] / \sqrt{2\pi\sigma_i^2}$$

iii. iterate for each data point → one «replica»



Constructing the replica

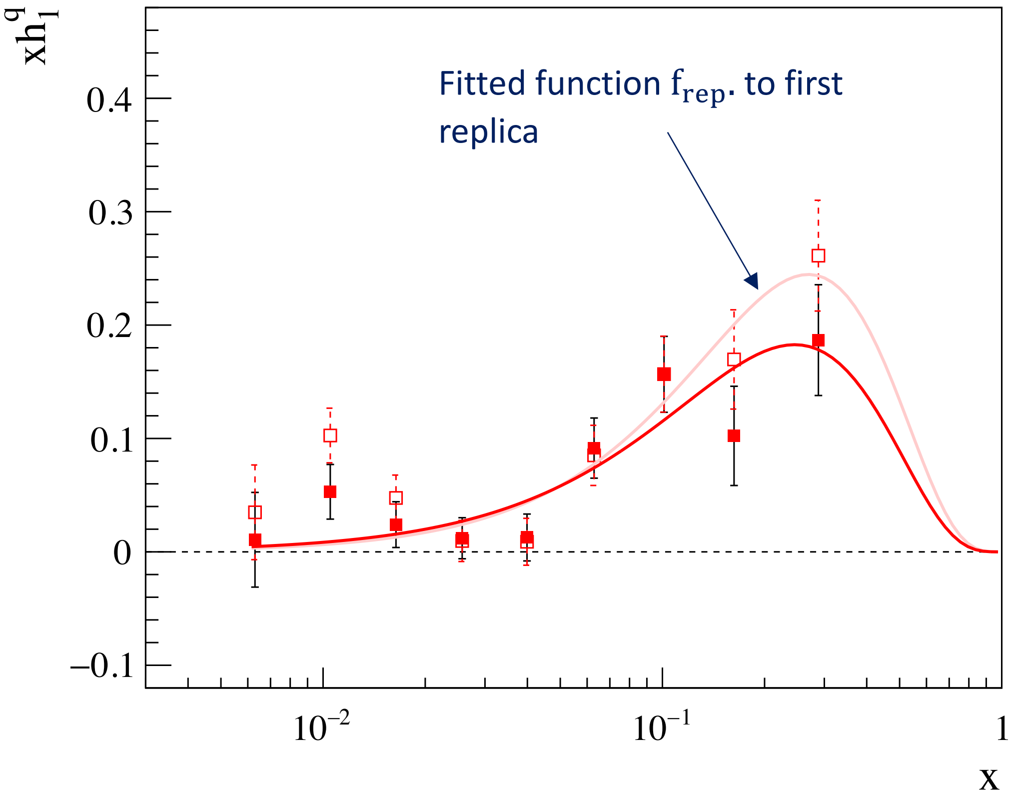
i. Consider a data point $y_i^{meas.}$ with statistical uncertainty σ_i

ii. Generate a new point y_i («pseudo-data») according to the gaussian distribution

$$N(y_i; y_i^{meas.}, \sigma_i) = \exp\left[-\frac{(y_i - y_i^{meas.})^2}{2\sigma_i^2}\right] / \sqrt{2\pi\sigma_i^2}$$

iii. iterate for each data point → one «replica»

iv. Fit of replica



Parameters of fit function fitted to data:

$\hat{a} \pm \sigma_{\hat{a}} = 3.47 \pm 1.52$

$\hat{b} \pm \sigma_{\hat{b}} = 1.31 \pm 0.19$

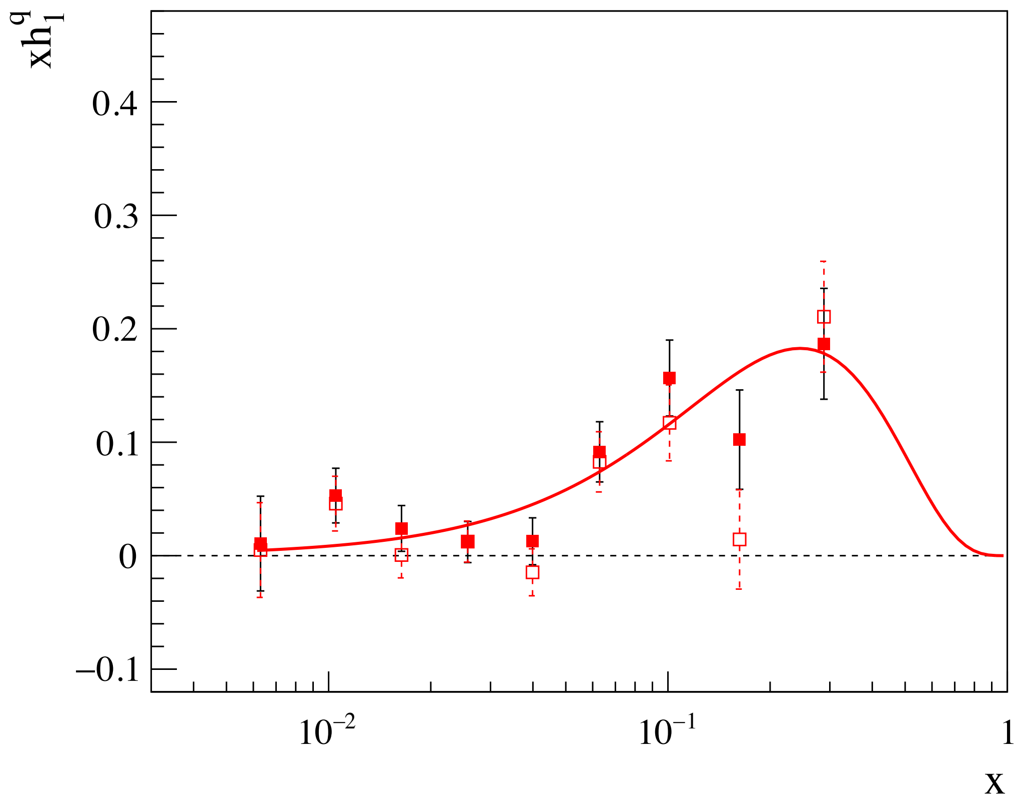
Parameters of fit function fitted to replica:

$\hat{a} \pm \sigma_{\hat{a}} = 3.22 \pm 1.44$

$\hat{b} \pm \sigma_{\hat{b}} = 1.26 \pm 0.19$

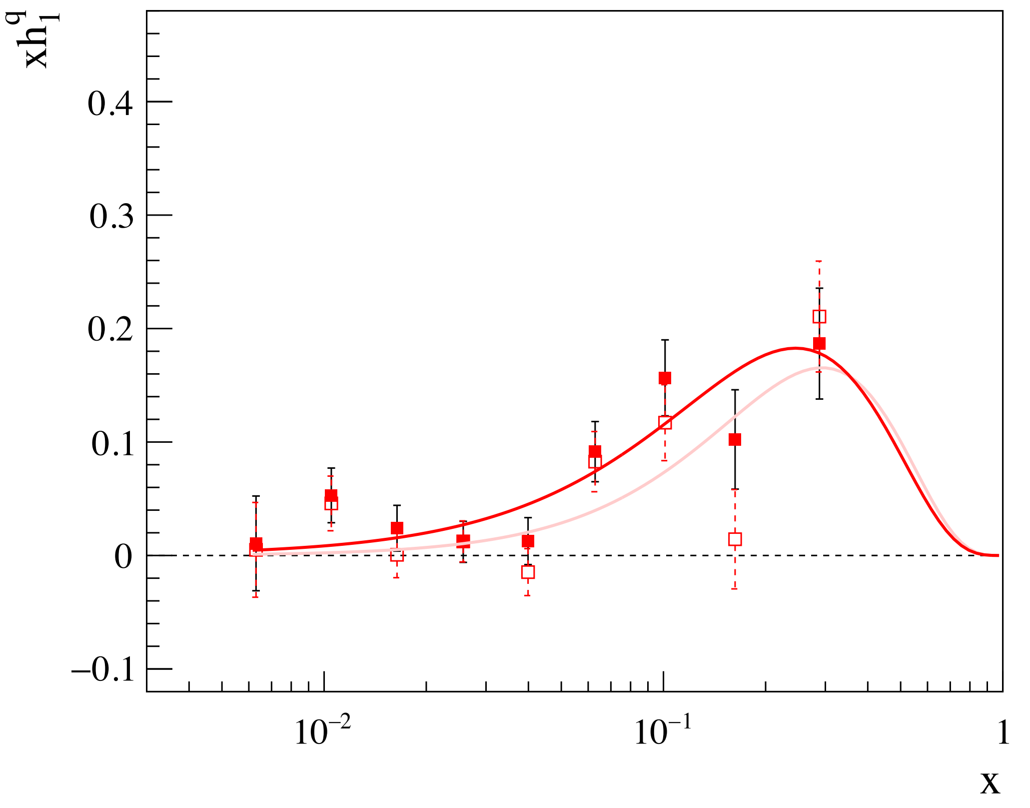
Constructing the replica

Generate a second replica and fit again..



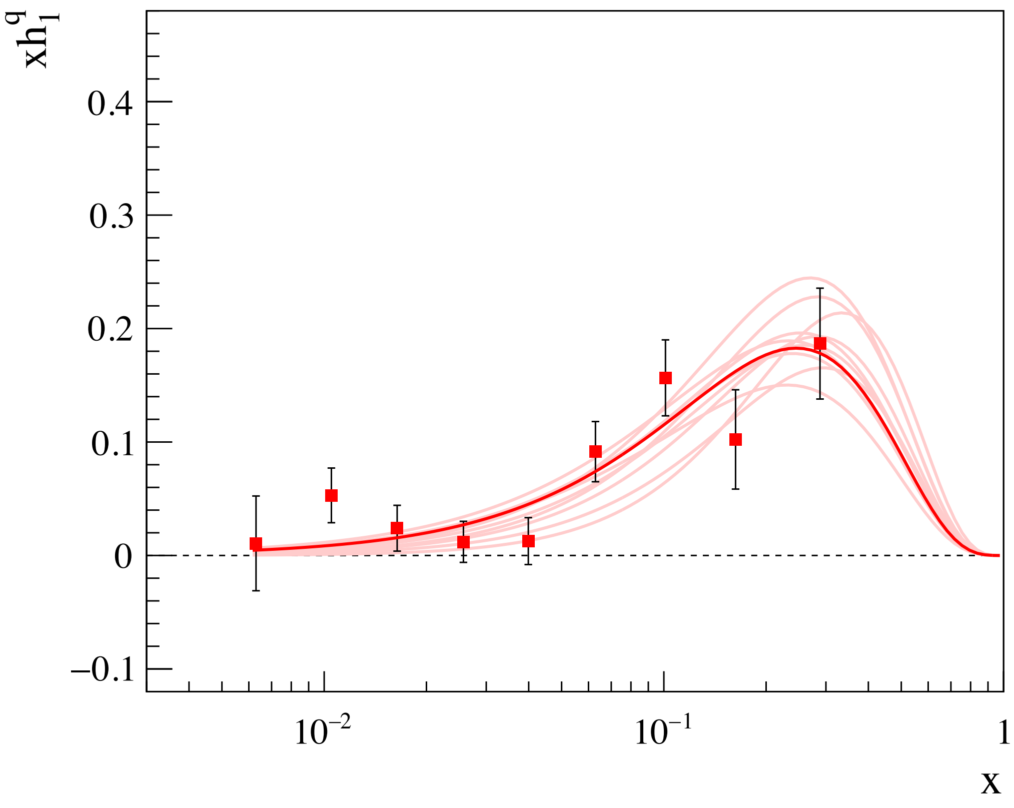
Constructing the replica

Generate a second replica and fit again..



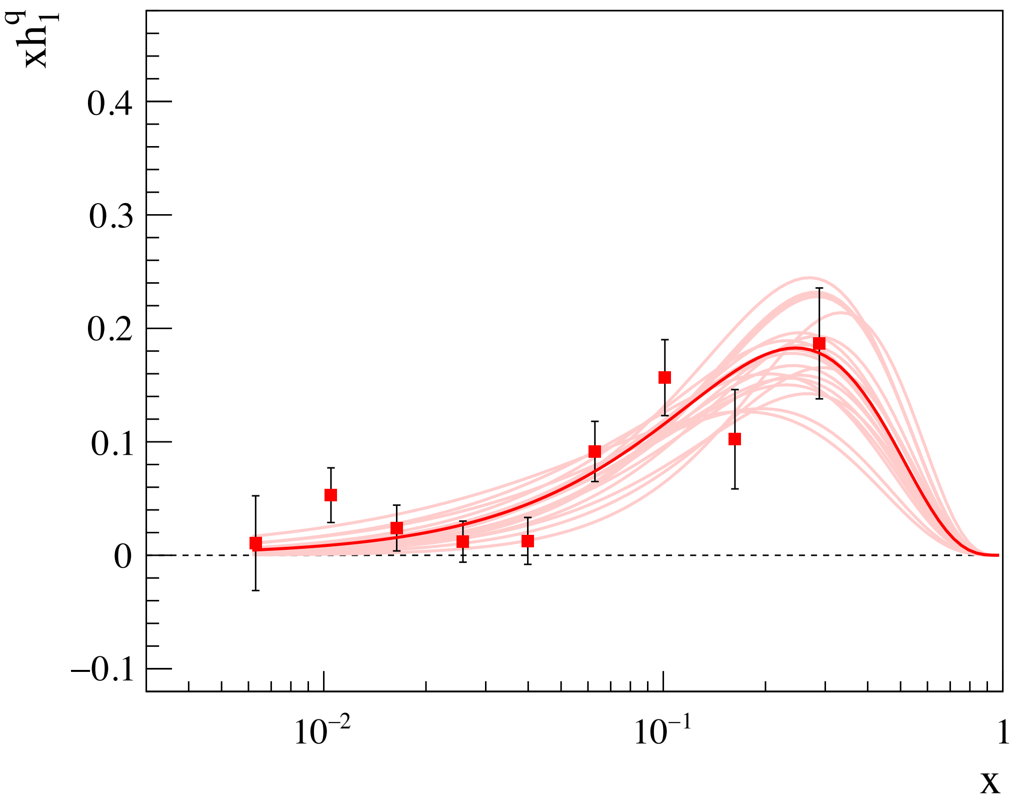
Constructing the replica

10 replica



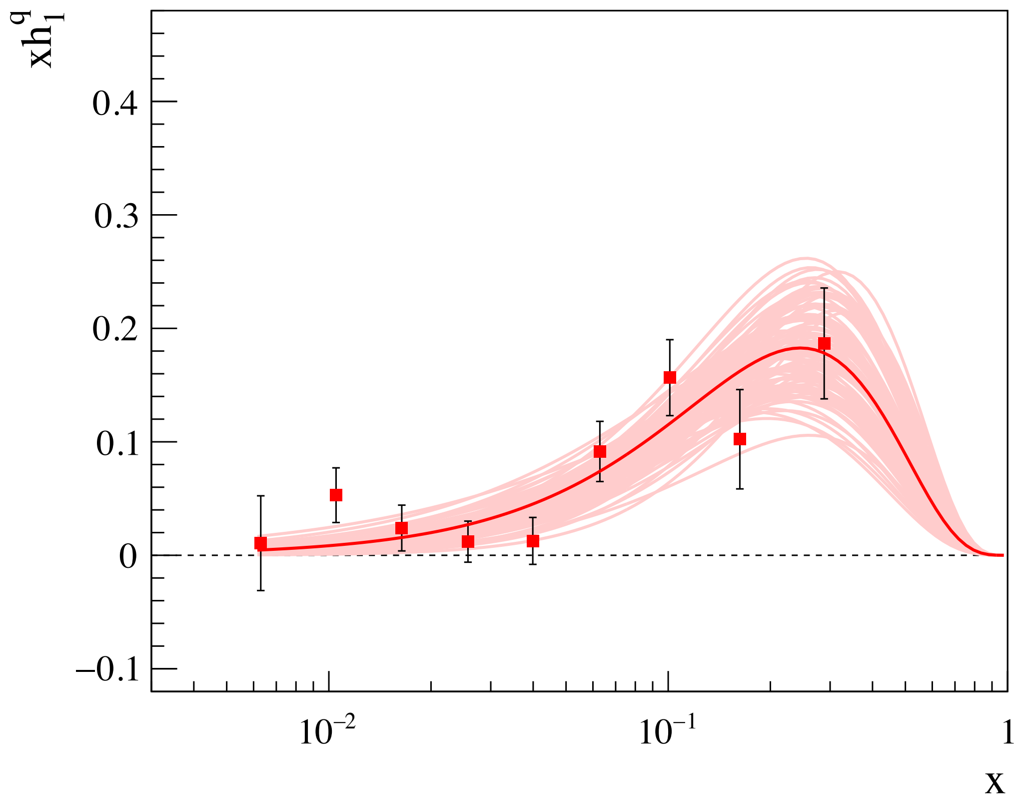
Constructing the replica

20 replica



Constructing the replica

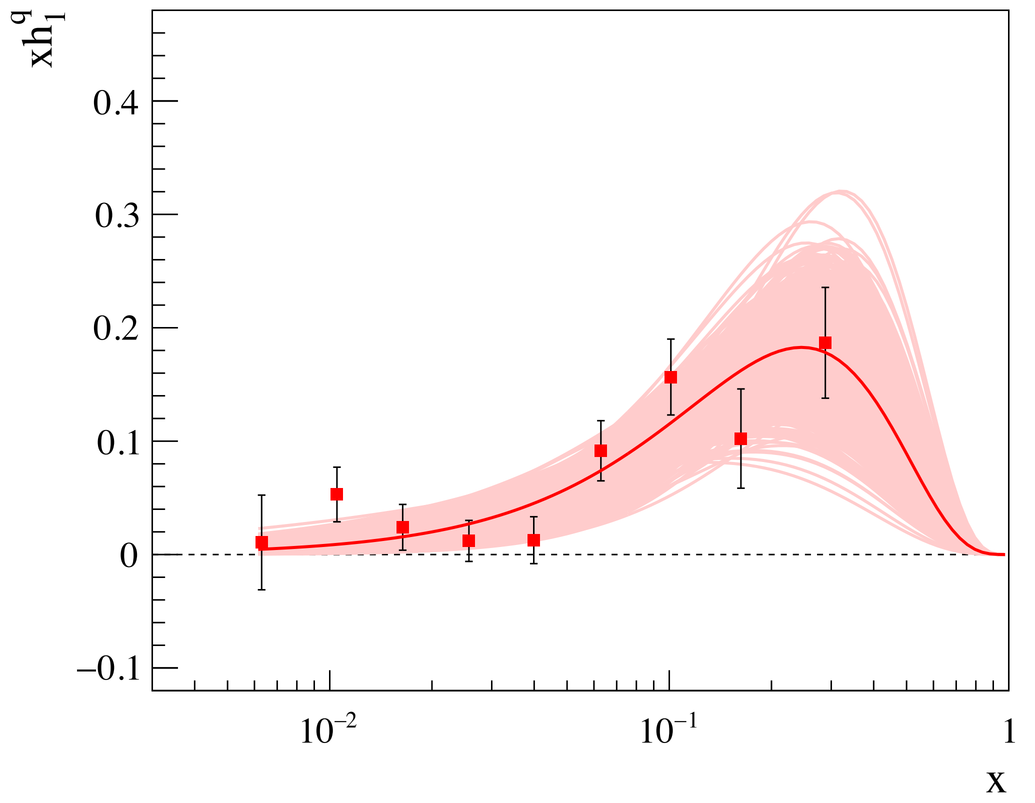
100 replica



Constructing the replica

1000 replica

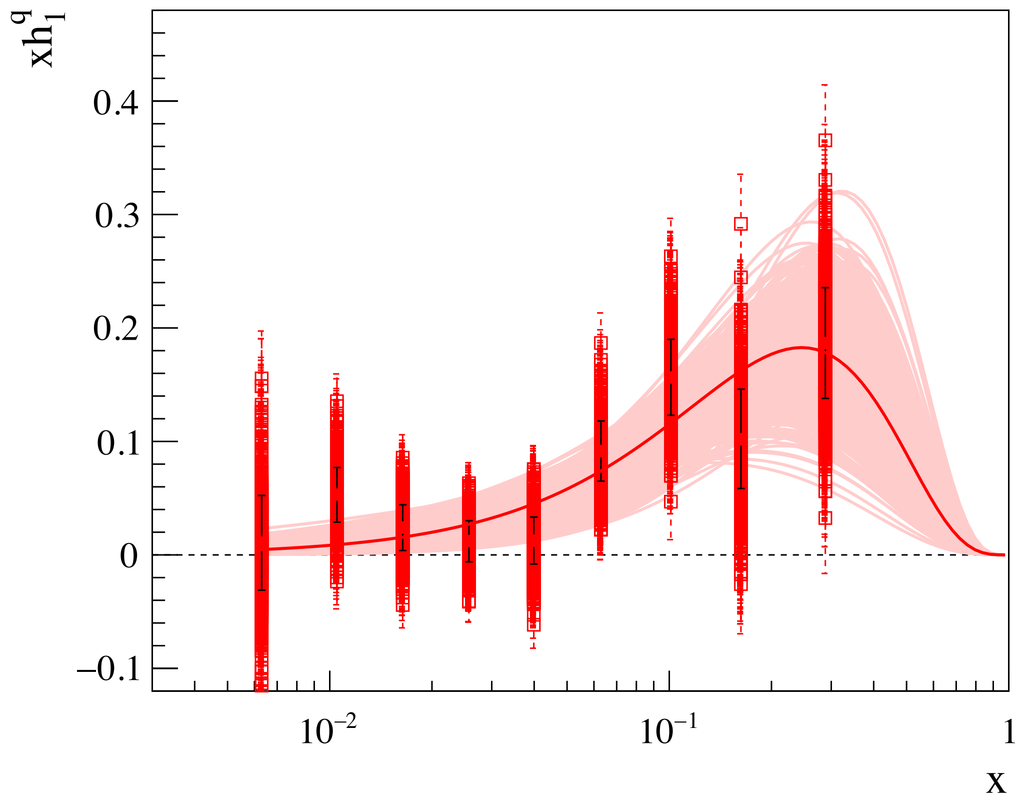
with real data applications it is difficult to generate more than 100-200 replica, due to the fact that a single fit could take hours..



Control step: distributions of the generated pseudodata

By construction, at each value of x the pseudodata should have a Gaussian distribution with

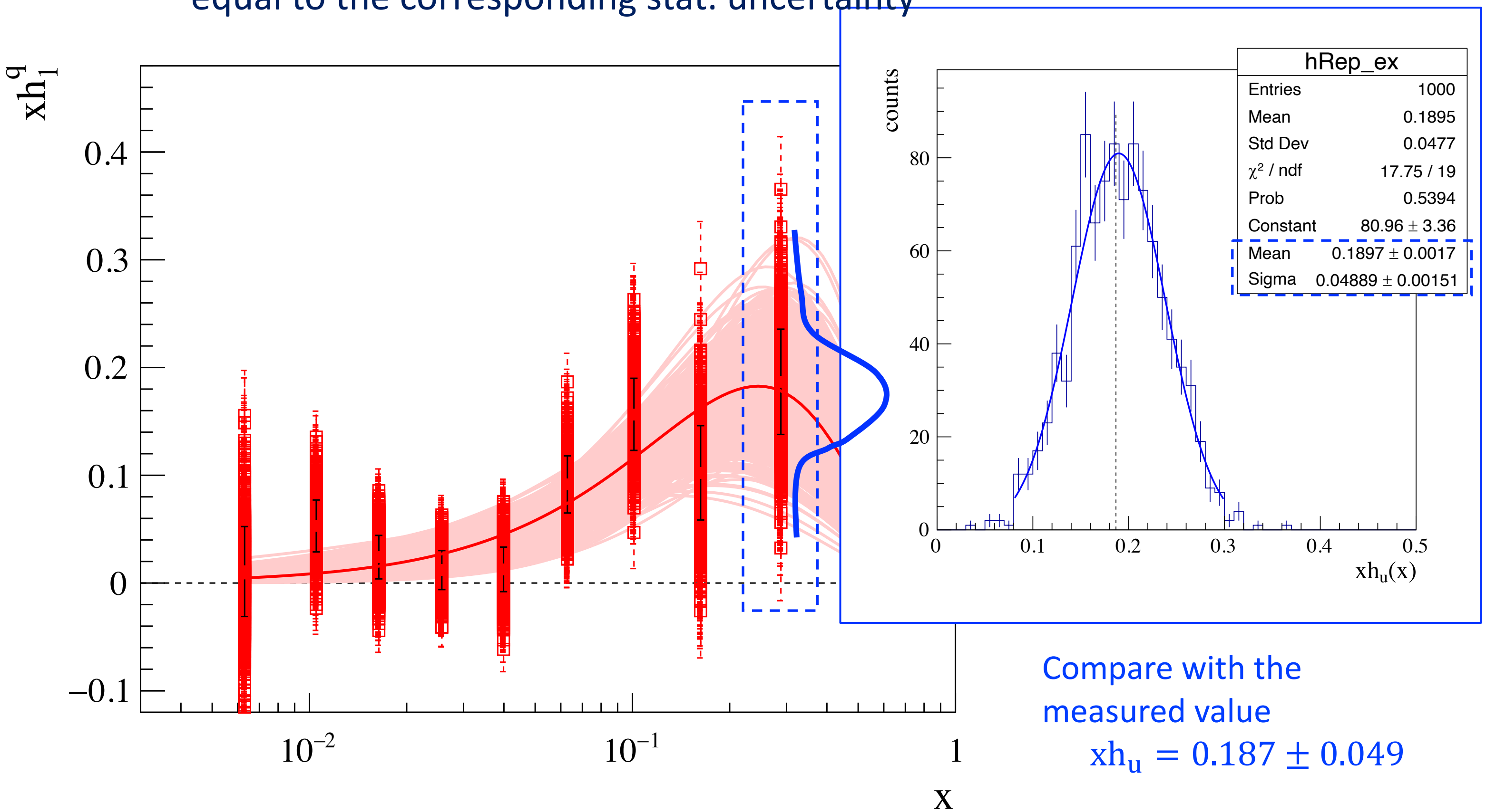
mean value μ	equal to the value of the data point
rms	equal to the corresponding stat. uncertainty



Control step: distributions of the generated pseudodata

By construction, at each value of x the pseudodata should have a Gaussian distribution with

mean value µ	equal to the value of the data point
rms	equal to the corresponding stat. uncertainty



Compare with the measured value

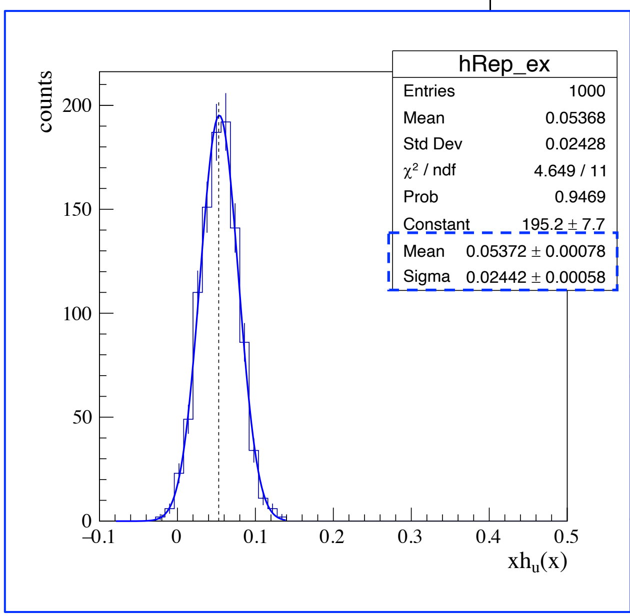
$xh_u = 0.187 \pm 0.049$

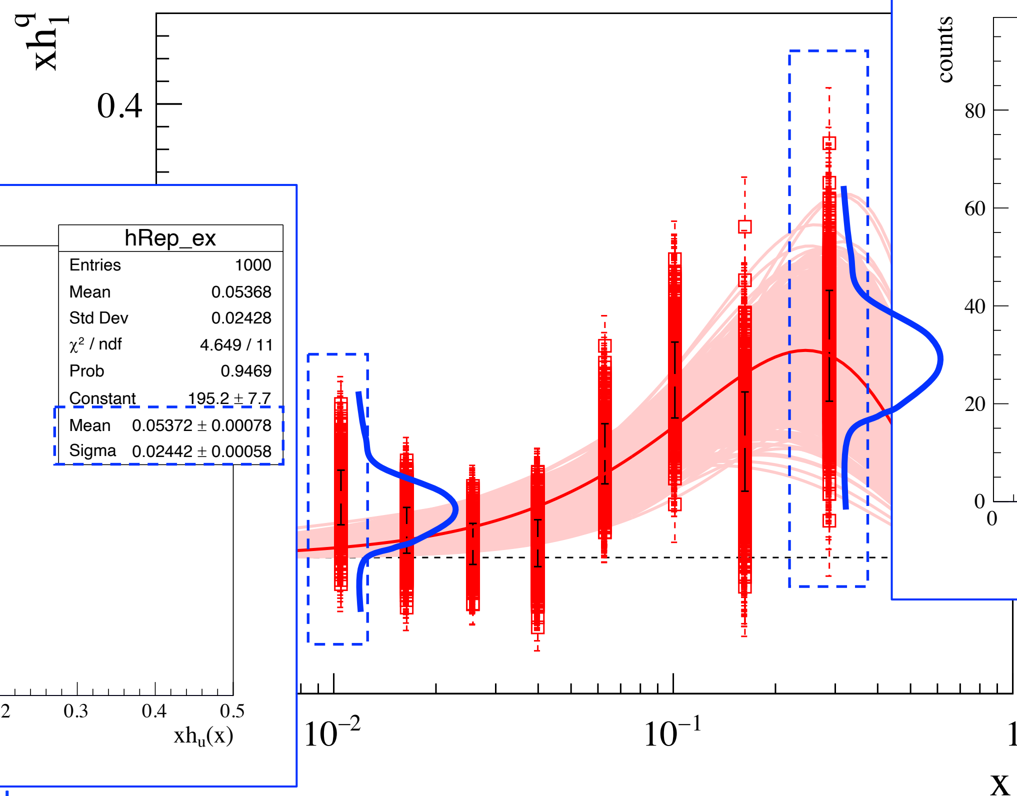
Control step: distributions of the generated pseudodata

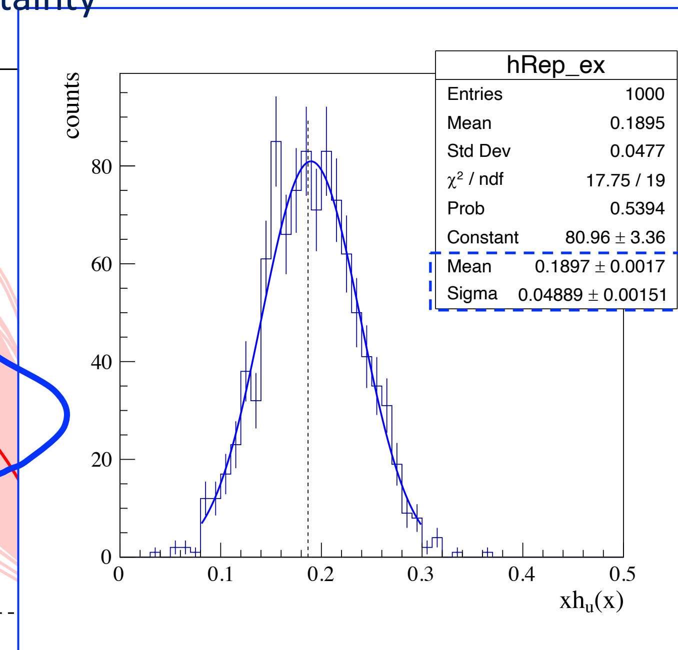
By construction, at each value of x the pseudodata should have a Gaussian distribution with

mean value μ equal to the value of the data point

rms equal to the corresponding stat. uncertainty







Compare with the measured value

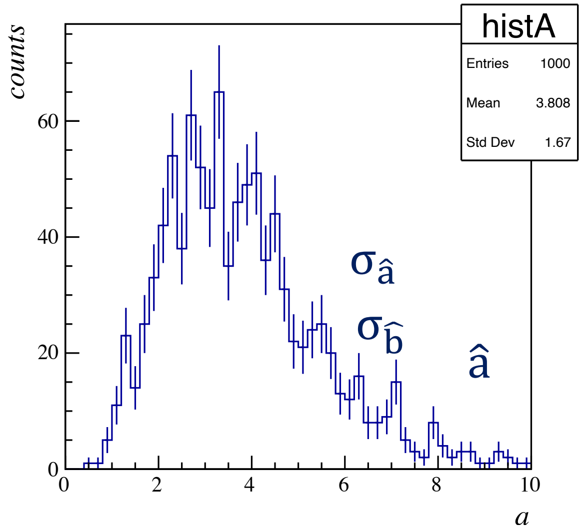
$xh_u = 0.187 \pm 0.049$

Compare with the measured value

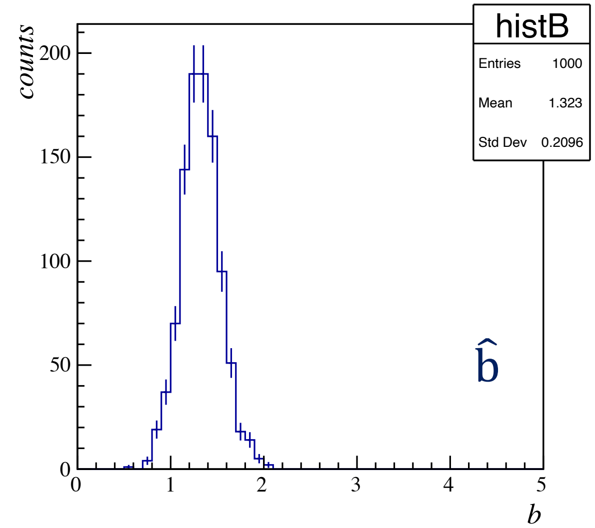
$xh_u = 0.053 \pm 0.024$

Distribution of fit parameters

Now we have the distributions of the parameters and their uncertainties



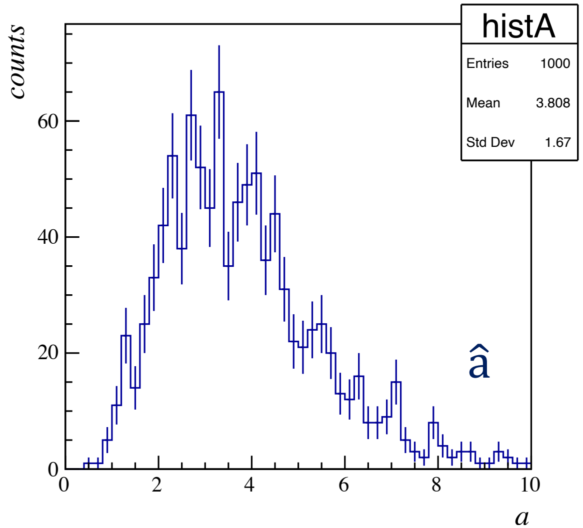
Distribution of $\hat{a}$ quite wide, as expected from the fit

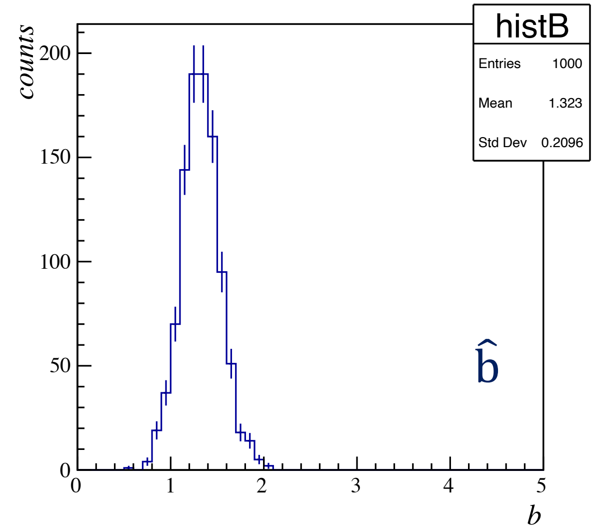


Distribution of $\hat{b}$ narrower, as expected from the fit

Distribution of fit parameters

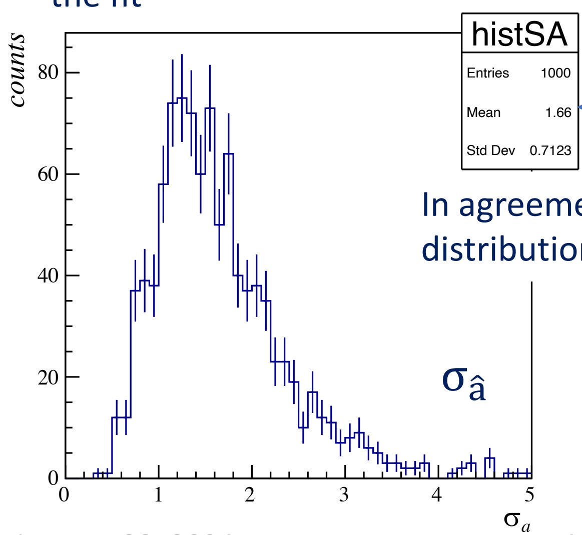
Now we have the distributions of the parameters and their uncertainties



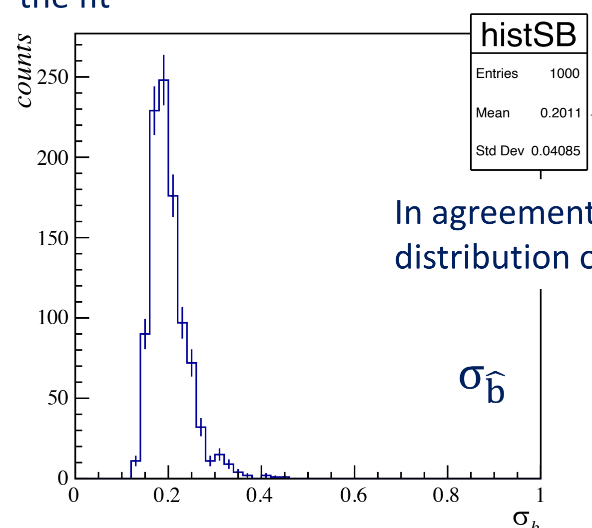


Distribution of â quite wide, as expected from the fit

Distribution of b̂ narrower, as expected from the fit



In agreement with the distribution of a

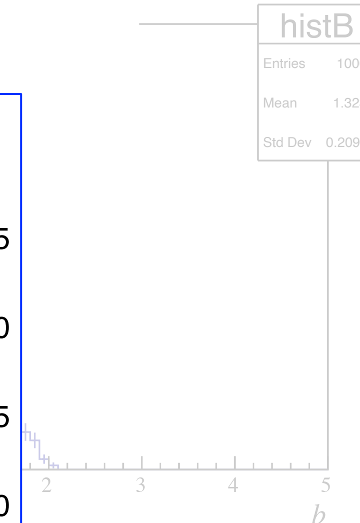


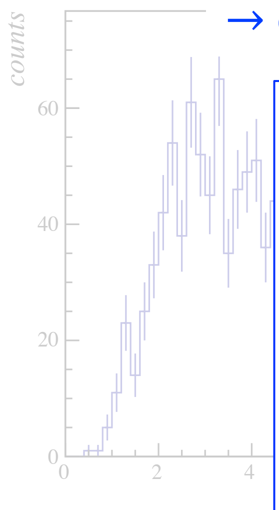
In agreement with the distribution of b

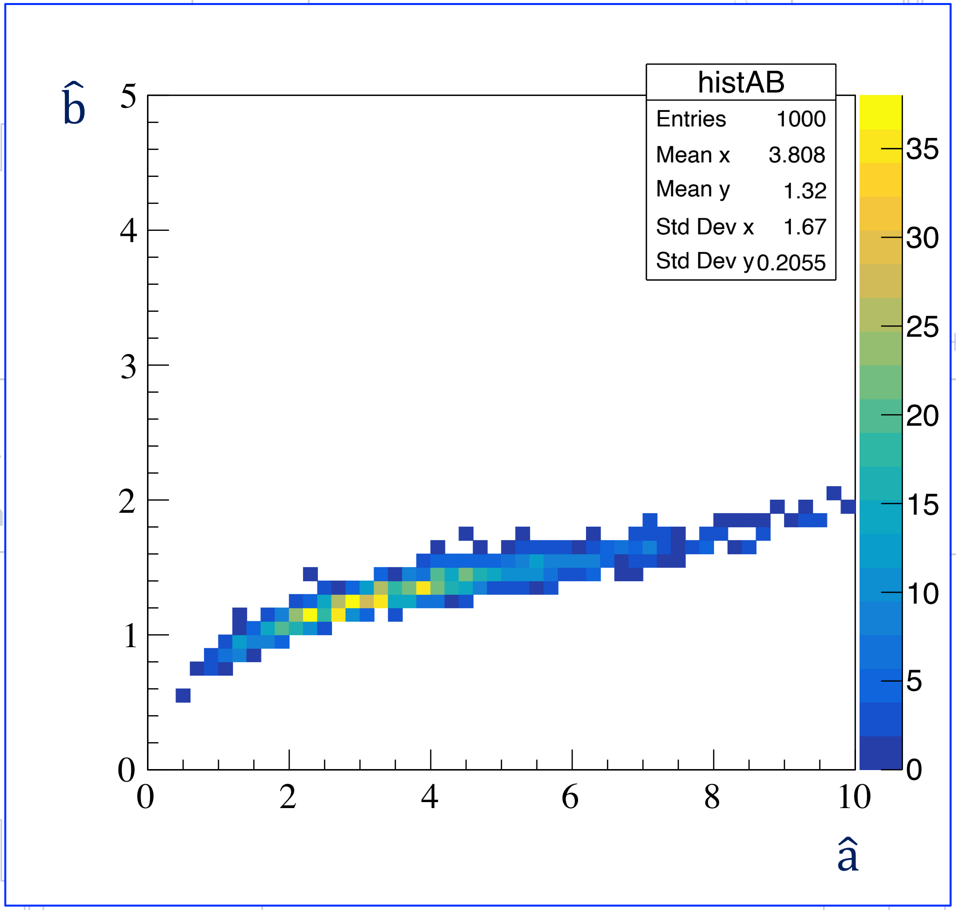
Distribution of fit parameters

Now we h We have access also to the joint distribution of a and b

→ correlations between the parameters

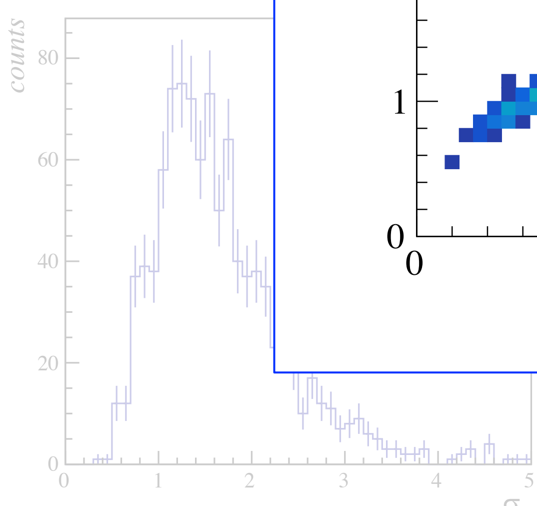


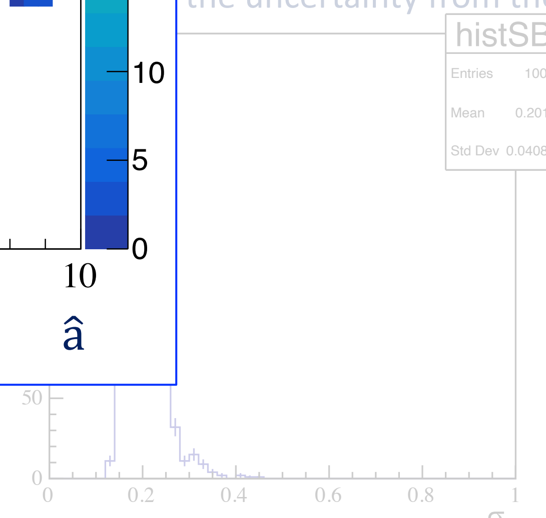




Distribution of ... of b̂ narrower..

expected by th ... the uncertainty from the initial fit



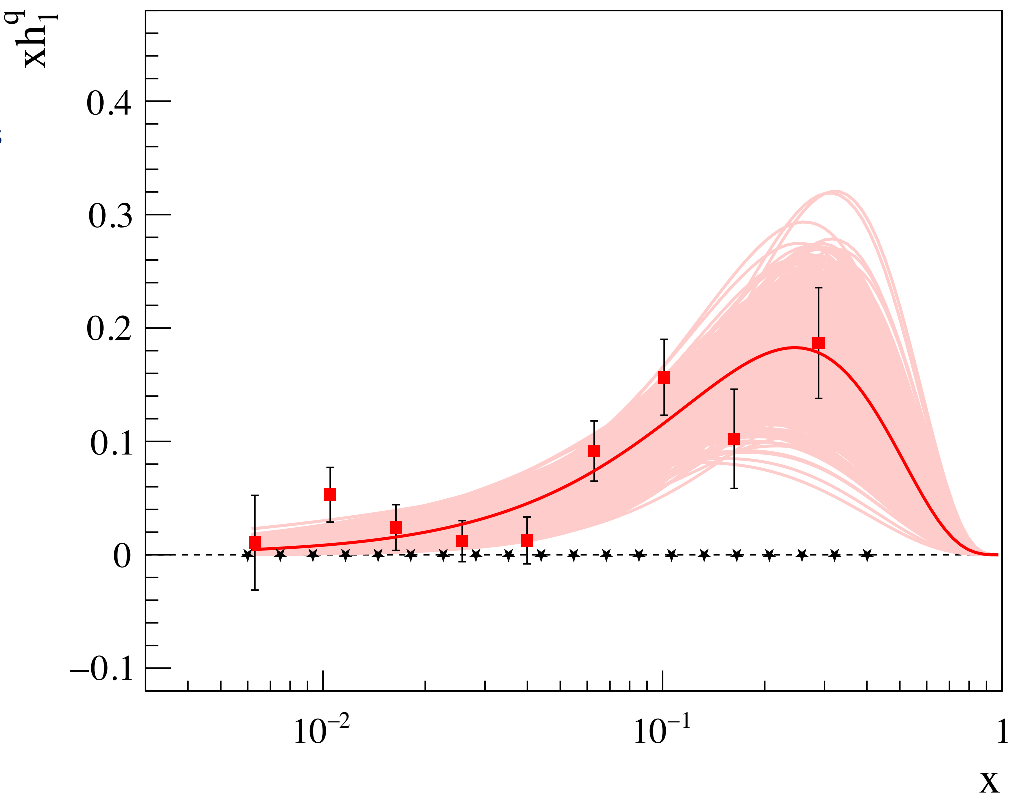


Constructing the uncertainty bands

Now we have all replica.. How can we construct the uncertainty bands at a given confidence level (CL)? here 68% and 90% CL

Look at the distribution of the value of the fit function in several x intervals

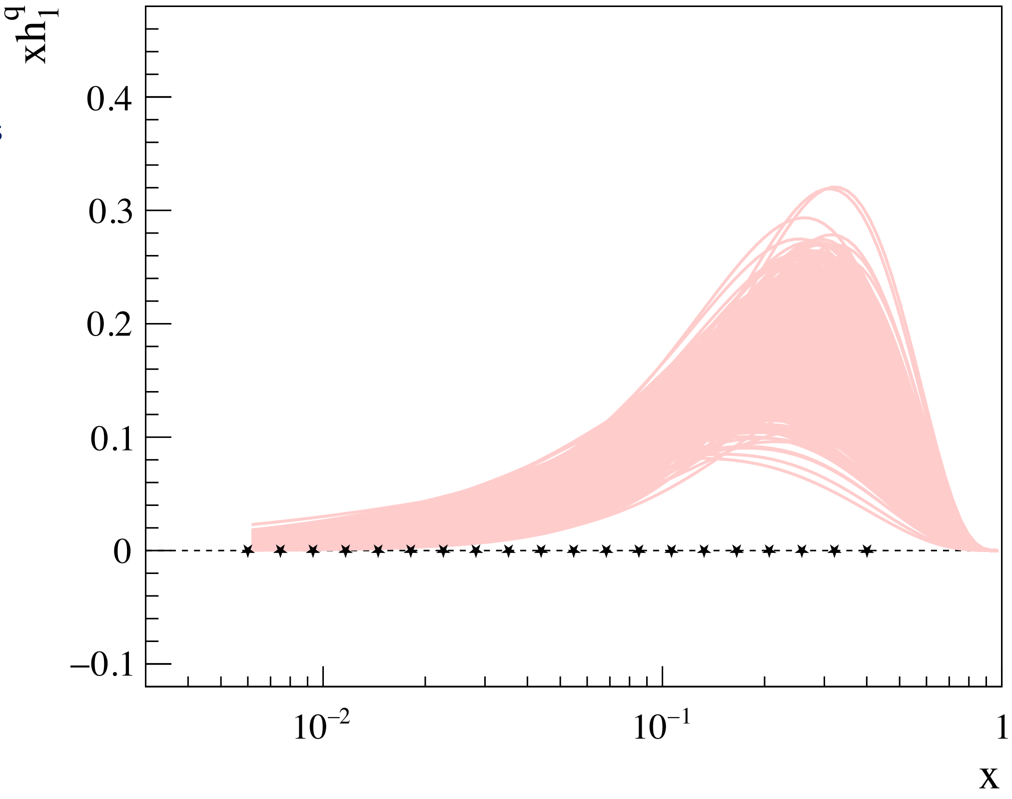
Steps:
Divide x-range in intervals



Constructing the uncertainty bands

For the moment, forget the data points and the fit function..
Concentrate on replica

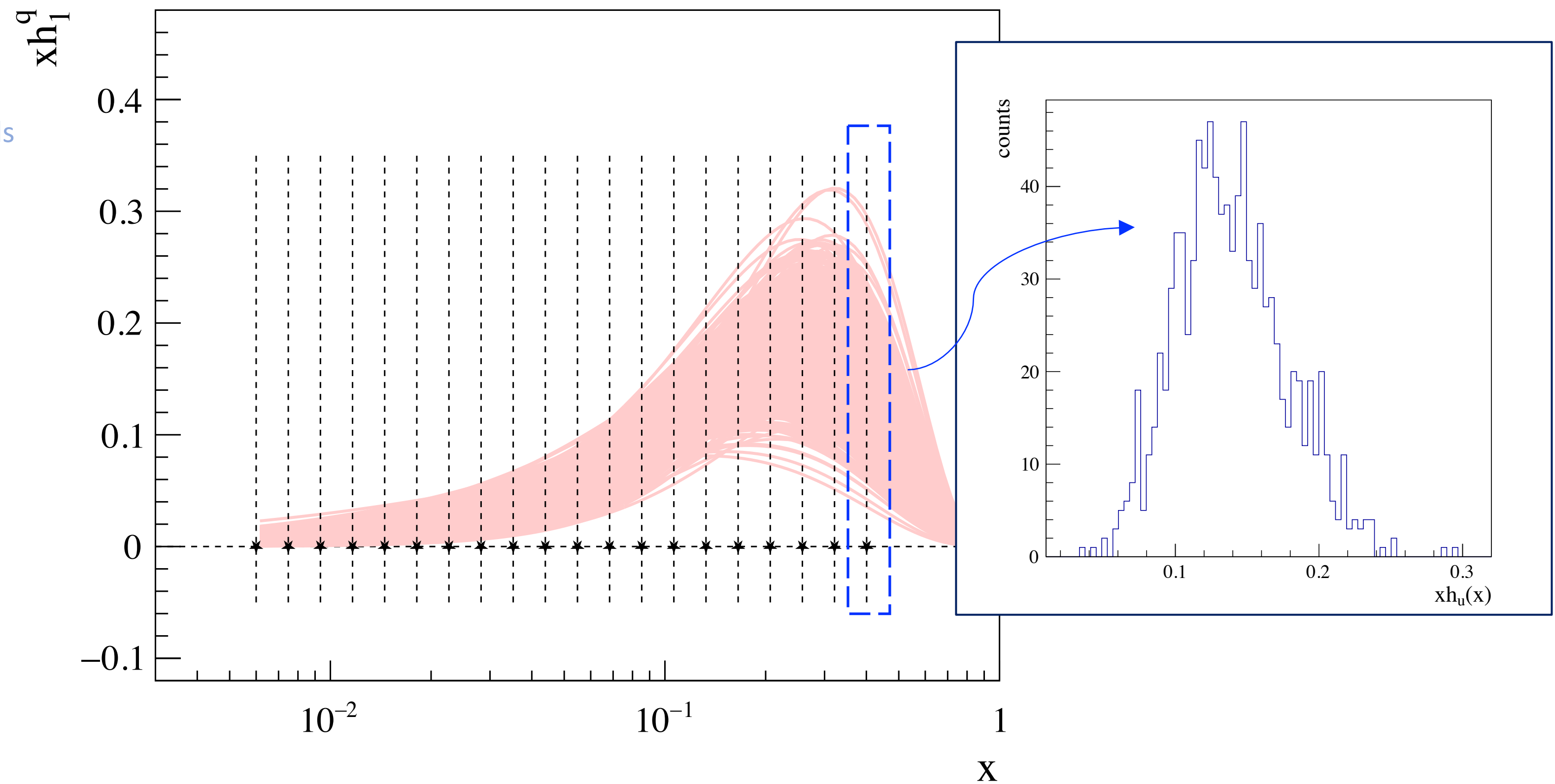
Steps:
Divide x-range in intervals



Constructing the uncertainty bands

For the moment, forget the data points and the fit function..
Concentrate on replica
and look at the distribution of the value of the fit function in each point

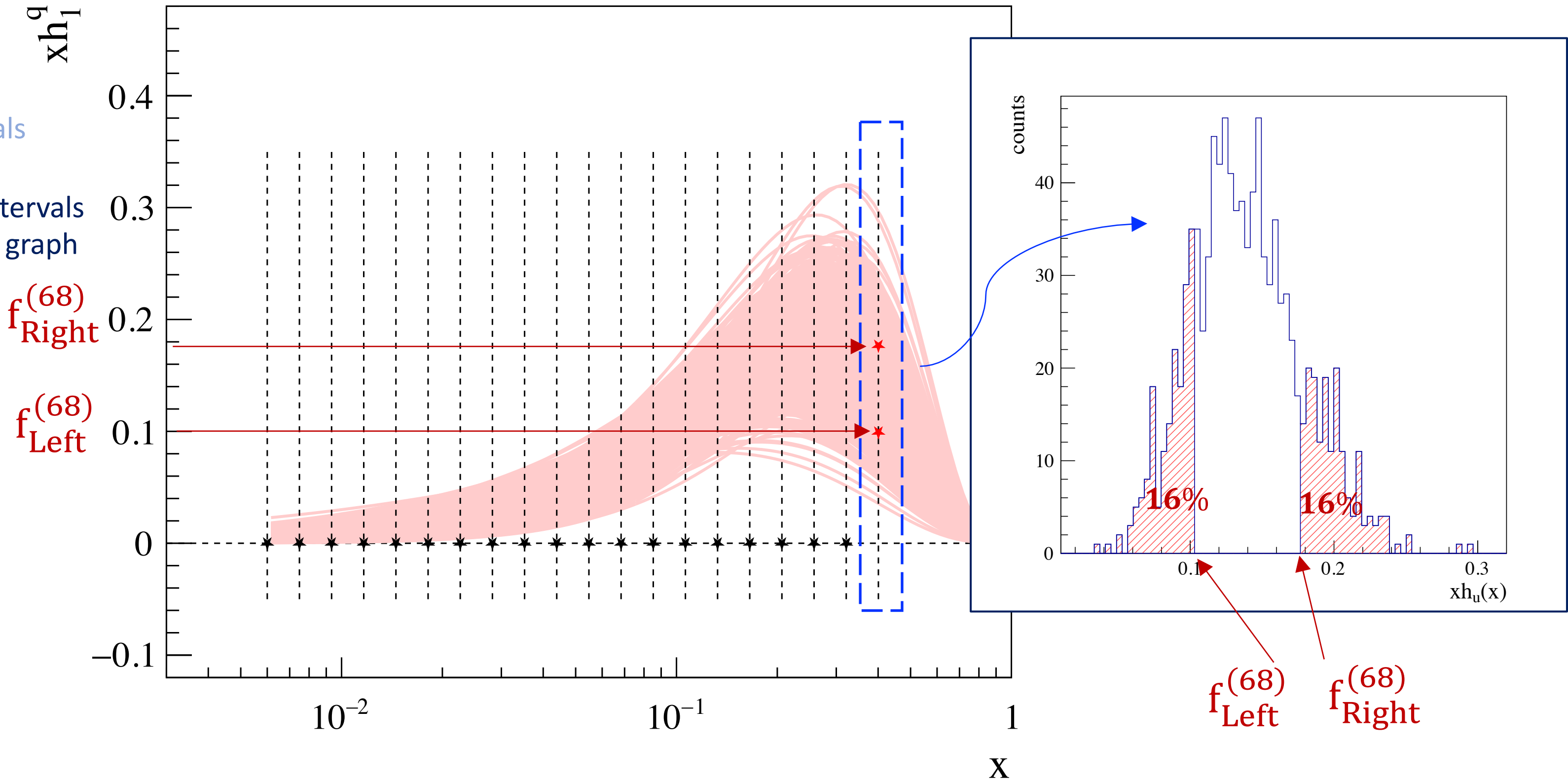
Steps:
Divide x-range in intervals
Distribution of $f_{rep}(x_i)$



Constructing the uncertainty bands

For the moment, forget the data points and the fit function..
Concentrate on replica
    and look at the distribution of the value of the fit function in each point

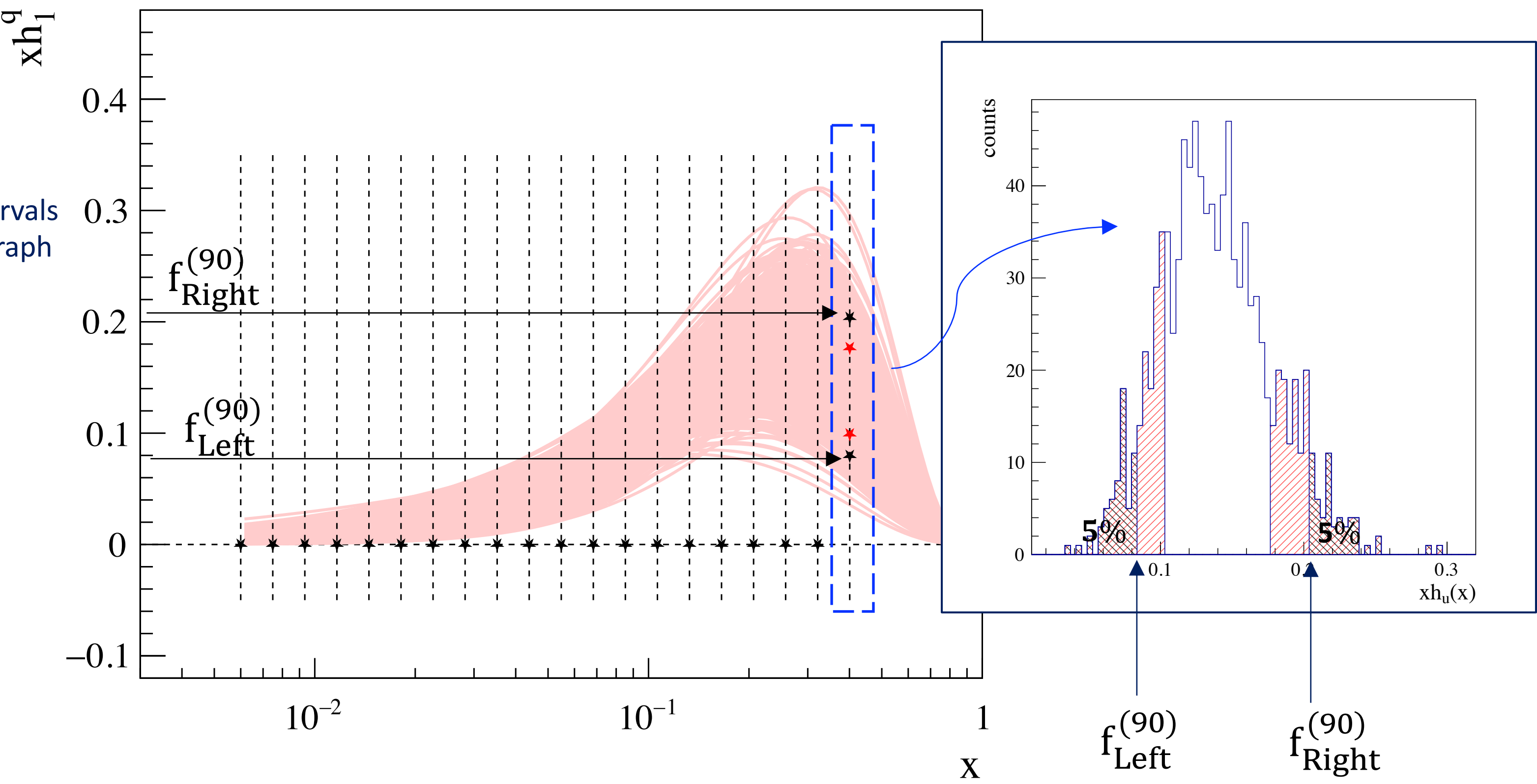
Steps:
Divide x-range in intervals
Distribution of $f_{rep}(x_i)$
Find 68% and 90% CL intervals and report point on the graph



Constructing the uncertainty bands

For the moment, forget the data points and the fit function..
Concentrate on replica
and look at the distribution of the value of the fit function in each point

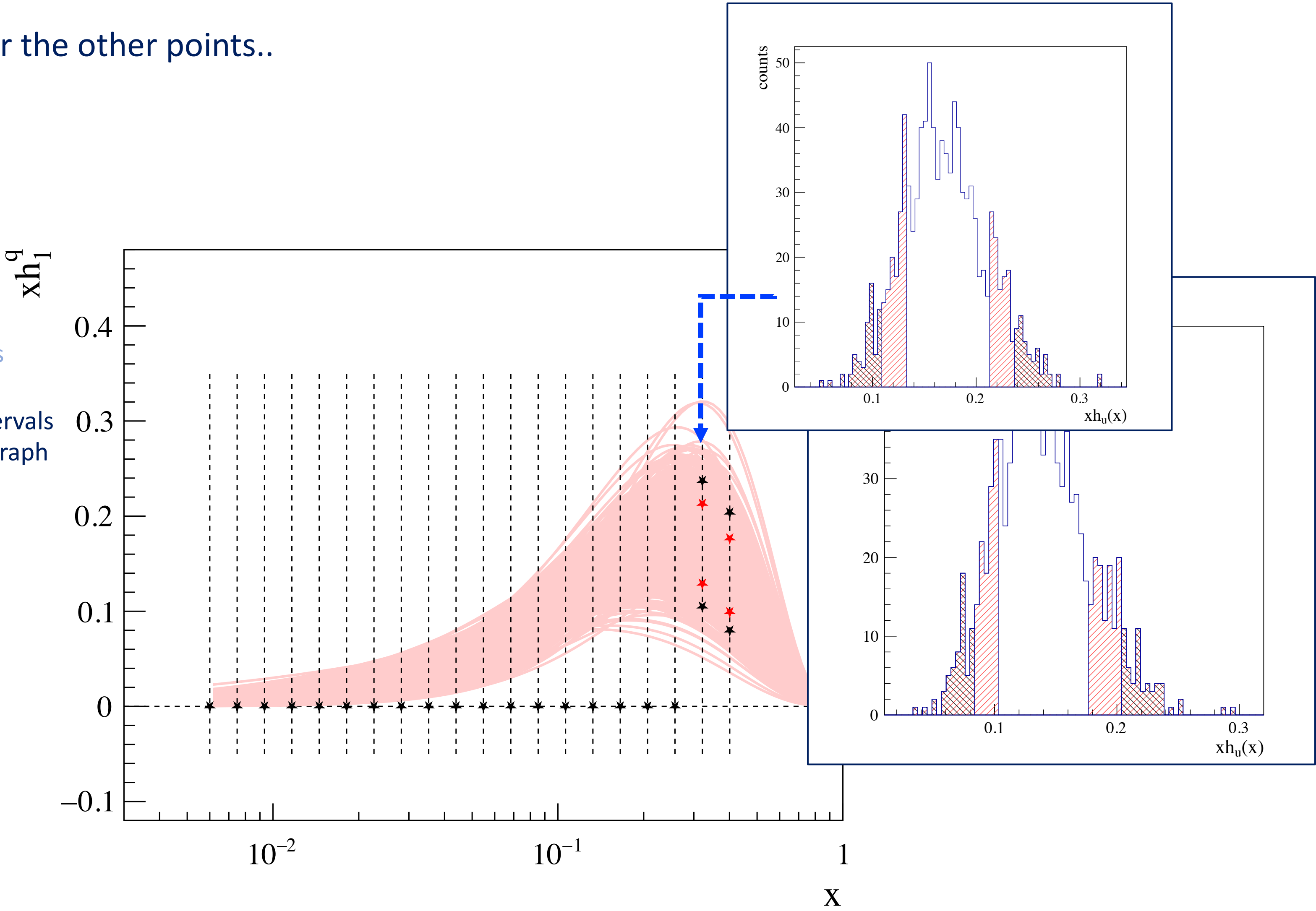
Steps:
Divide x-range in intervals
Distribution of $f_{rep}(x_i)$
Find 68% and 90% CL intervals and report point on the graph



Constructing the uncertainty bands

Repeat for the other points..

Steps:
Divide x-range in intervals
Distribution of $f_{rep}(x_i)$
Find 68% and 90% CL intervals and report point on the graph



Constructing the uncertainty bands

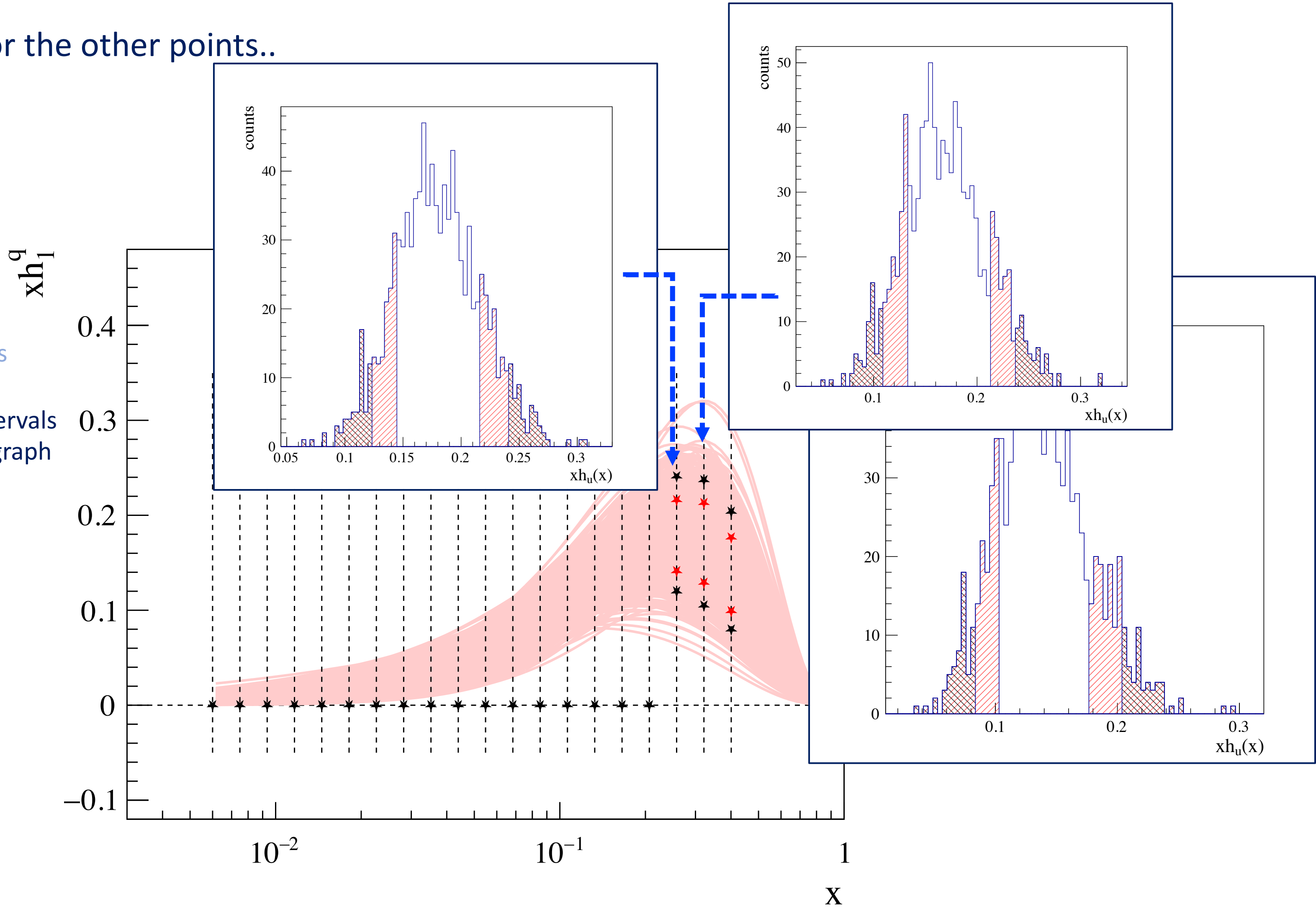
Repeat for the other points..

Steps:

Divide x-range in intervals

Distribution of $f_{\mathrm{rep}}(x_i)$

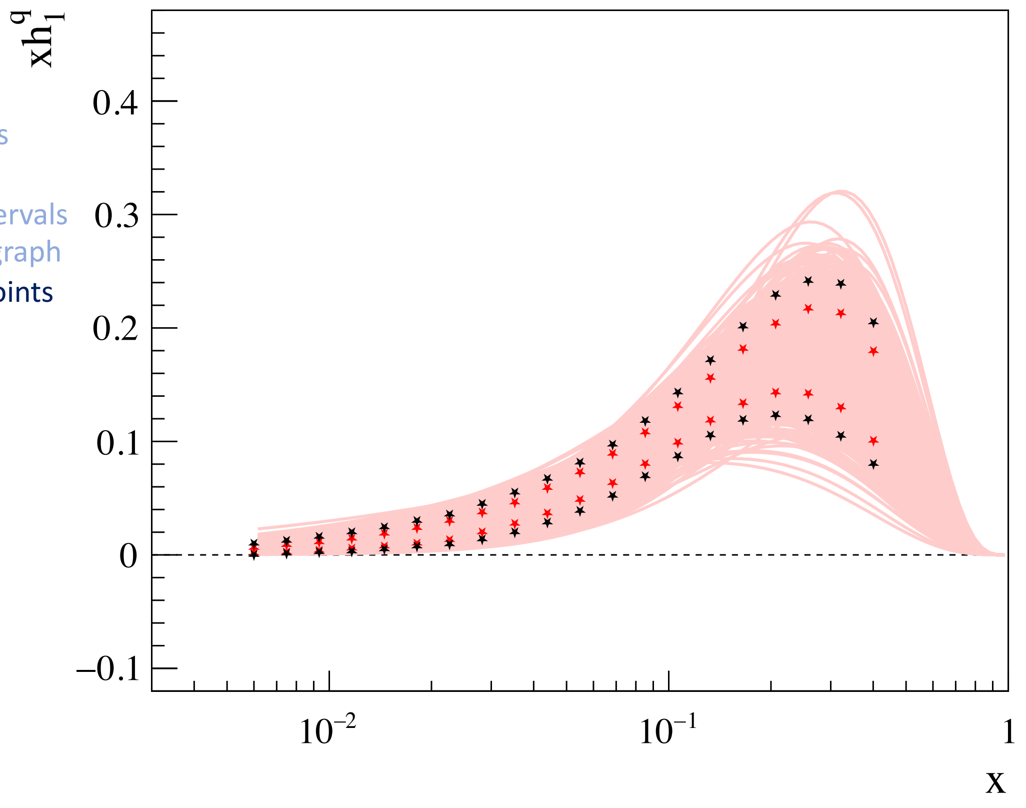
Find 68% and 90% CL intervals and report point on the graph



Constructing the uncertainty bands

Repeat for the other points..

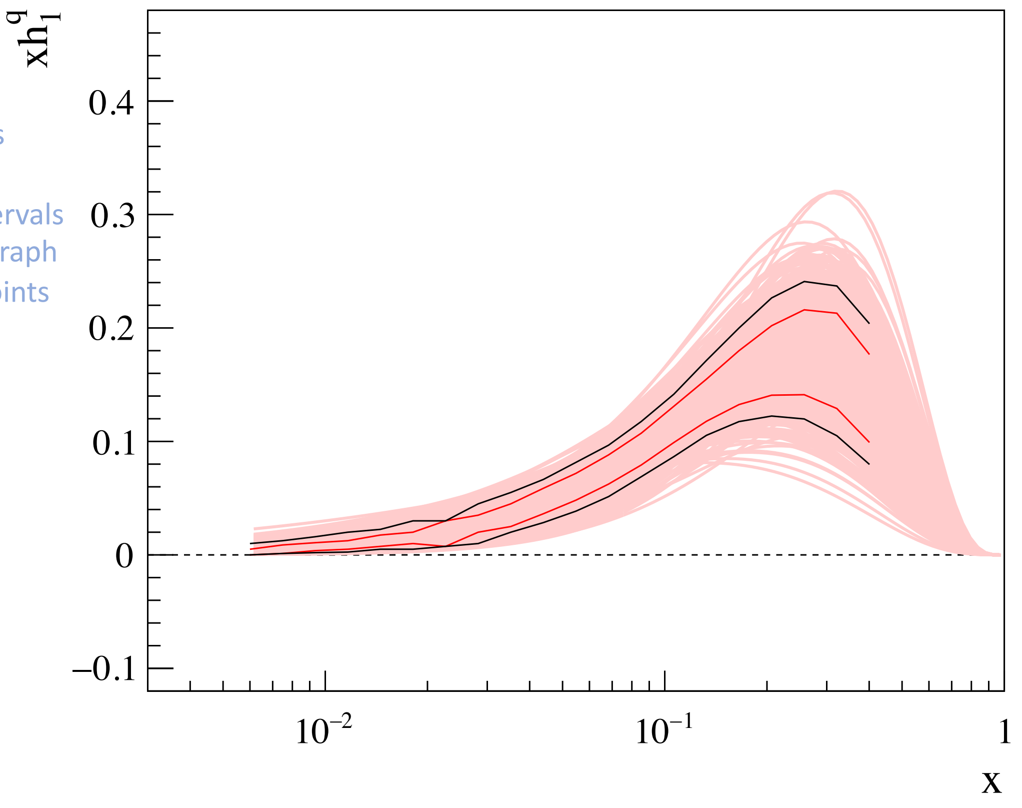
Steps:
Divide x-range in intervals
Distribution of $f_{rep}(x_i)$
Find 68% and 90% CL intervals and report point on the graph
Iterate for all chosen x-points



Constructing the uncertainty bands

Join the points with lines..

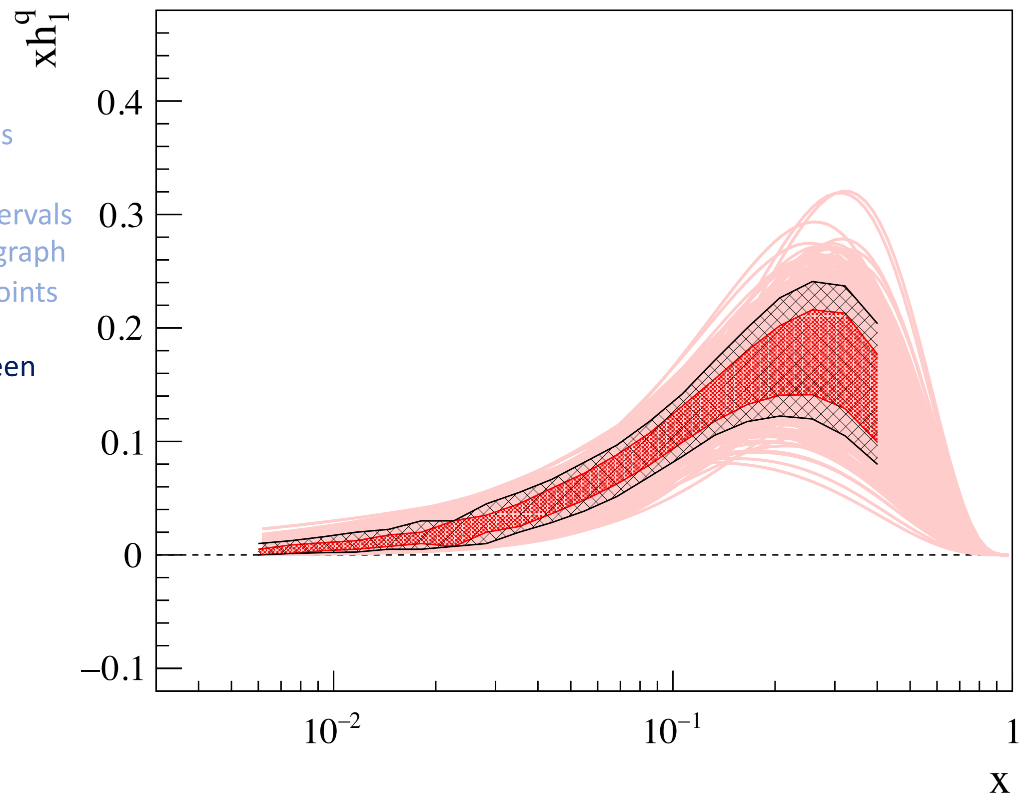
Steps:
Divide x-range in intervals
Distribution of $f_{rep}(x_i)$
Find 68% and 90% CL intervals and report point on the graph
Iterate for all chosen x-points
Join points with lines



Constructing the uncertainty bands

Join the points with lines.. and fill area in between the lines
→ now we have (uncertainty) bands!!

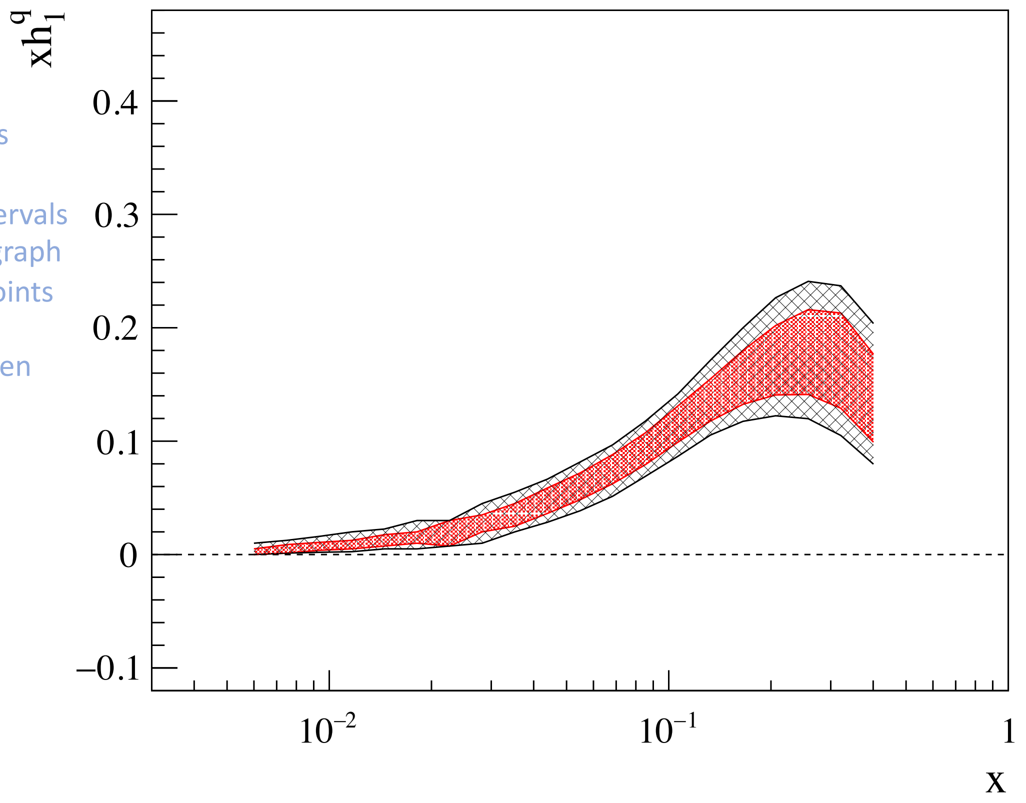
Steps:
Divide x-range in intervals
Distribution of $f_{rep}(x_i)$
Find 68% and 90% CL intervals and report point on the graph
Iterate for all chosen x-points
Join points with lines..
and fill the area in between



Constructing the uncertainty bands

Finally, remove all the fit functions of the replica and draw the data points!
Now we are done!

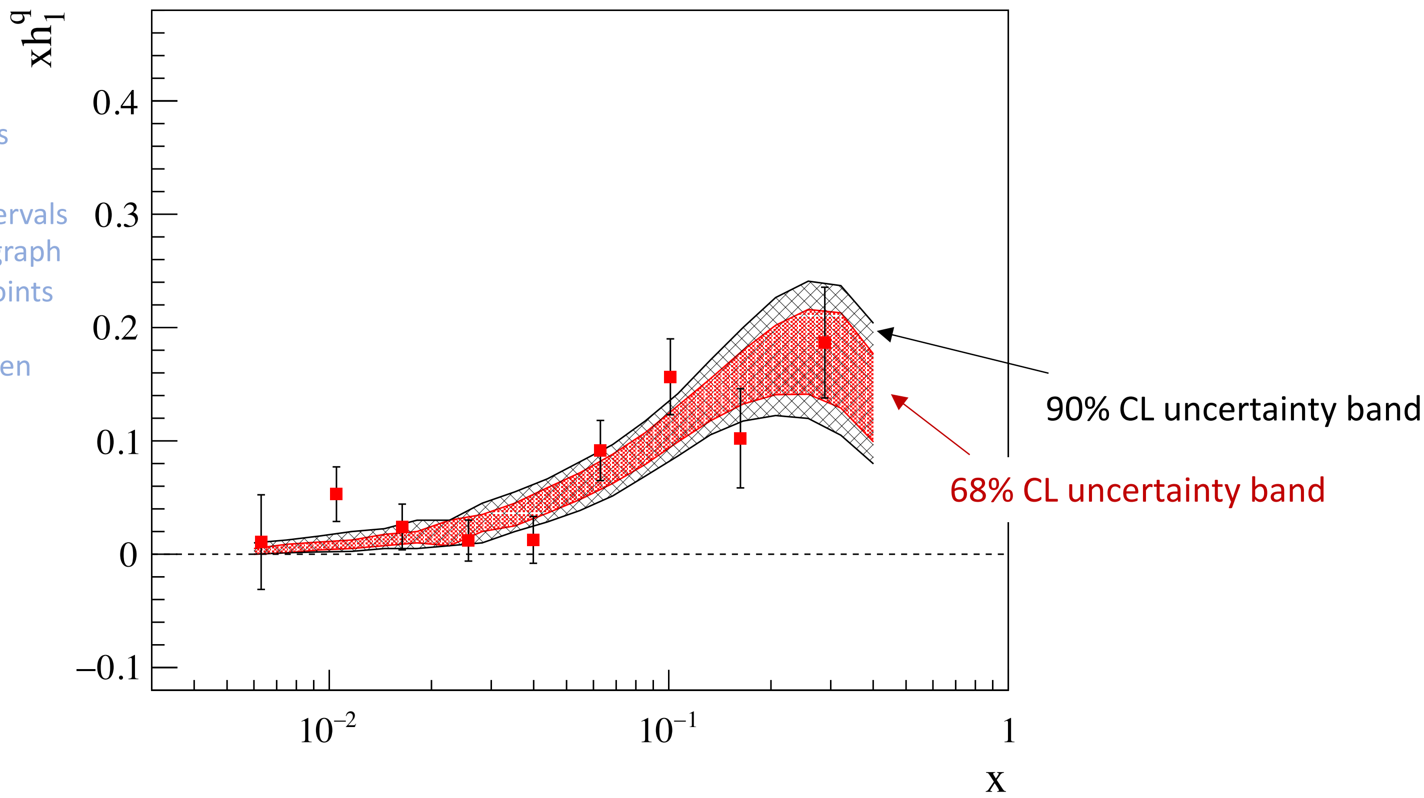
Steps:
Divide x-range in intervals
Distribution of $f_{rep}(x_i)$
Find 68% and 90% CL intervals and report point on the graph
Iterate for all chosen x-points
Join points with lines..
and fill the area in between
Remove replica



Constructing the uncertainty bands

Finally, remove all the fit functions of the replica and draw the data points!
Now we are done !

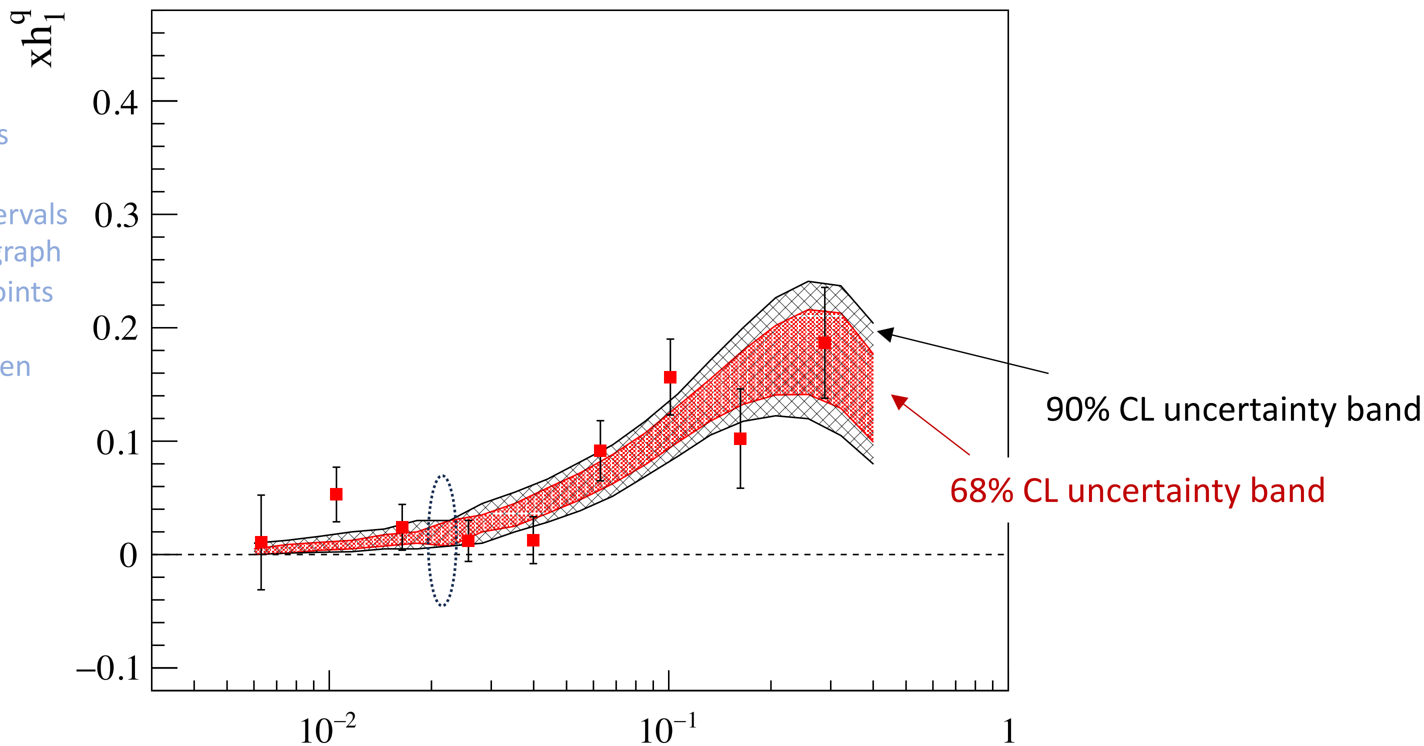
Steps:
Divide x-range in intervals
Distribution of $f_{rep}(x_i)$
Find 68% and 90% CL intervals and report point on the graph
Iterate for all chosen x-points
Join points with lines..
and fill the area in between
Remove replica
Draw data points



Constructing the uncertainty bands

Finally, remove all the fit functions of the replica and draw the data points!
Now we are done !

Steps:
Divide x-range in intervals
Distribution of $f_{rep}(x_i)$
Find 68% and 90% CL intervals and report point on the graph
Iterate for all chosen x-points
Join points with lines..
and fill the area in between
Remove replica
Draw data points

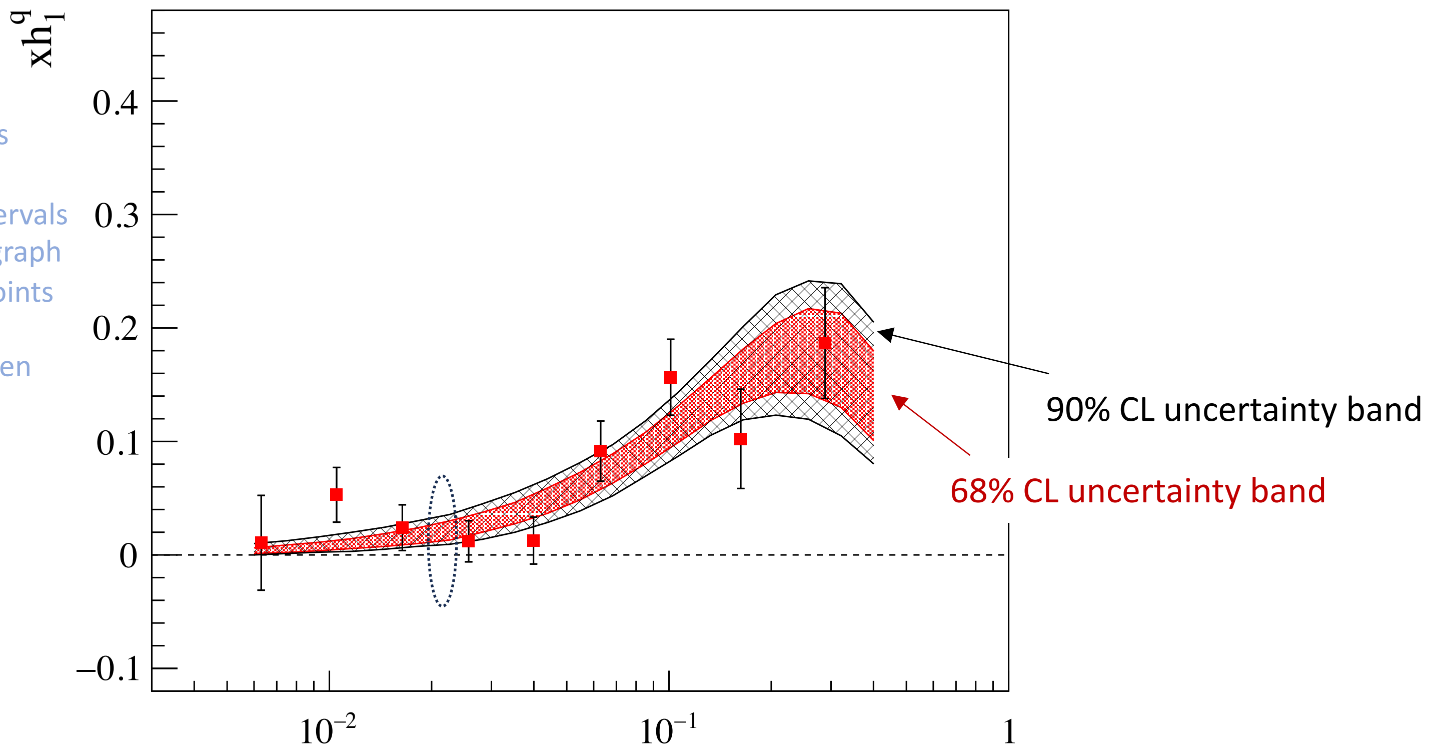


Of course a bit of tuning is always needed..

Constructing the uncertainty bands

Finally, remove all the fit functions of the replica and draw the data points!
Now we are done !

Steps:
Divide x-range in intervals
Distribution of $f_{rep}(x_i)$
Find 68% and 90% CL intervals and report point on the graph
Iterate for all chosen x-points
Join points with lines..
and fill the area in between
Remove replica
Draw data points



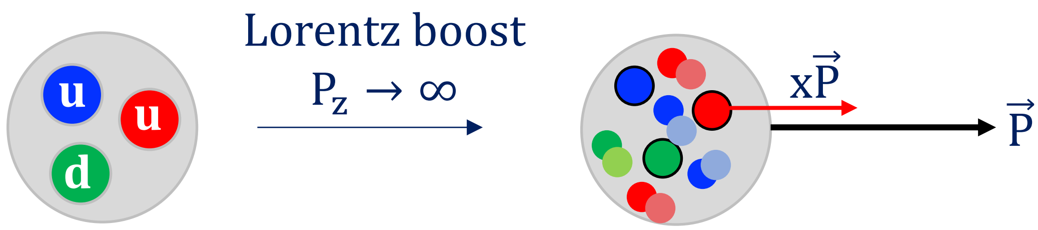
Of course a bit of tuning is always needed..
Ex. more precise calculation of 68% and 90% CL points for each histogram (increase number of bins)

What are the data points we played with?

They represent a "property” of quarks in a proton..

What are the data points we played with?

They represent a "property" of quarks in a proton..



proton at rest

proton with $v \simeq 1$

collection of «partons» (quarks and gluons)

valence and «sea» partons

Feynman '68

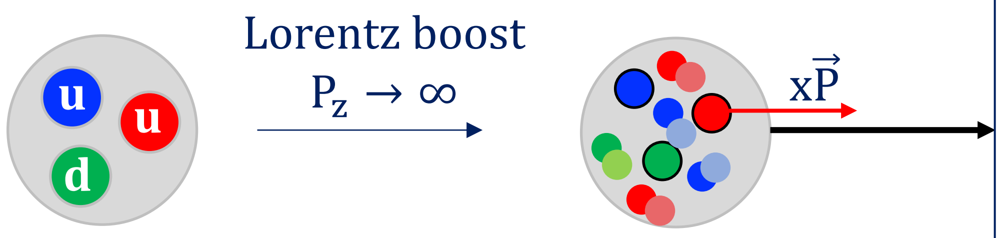
Distribution of partons described by the

parton distribution function $\mathbf{f_1^q(x)}$

x is the fraction of the nucleon's momentum
carried by quark (parton) q
(known as the «Bjorken variable»)

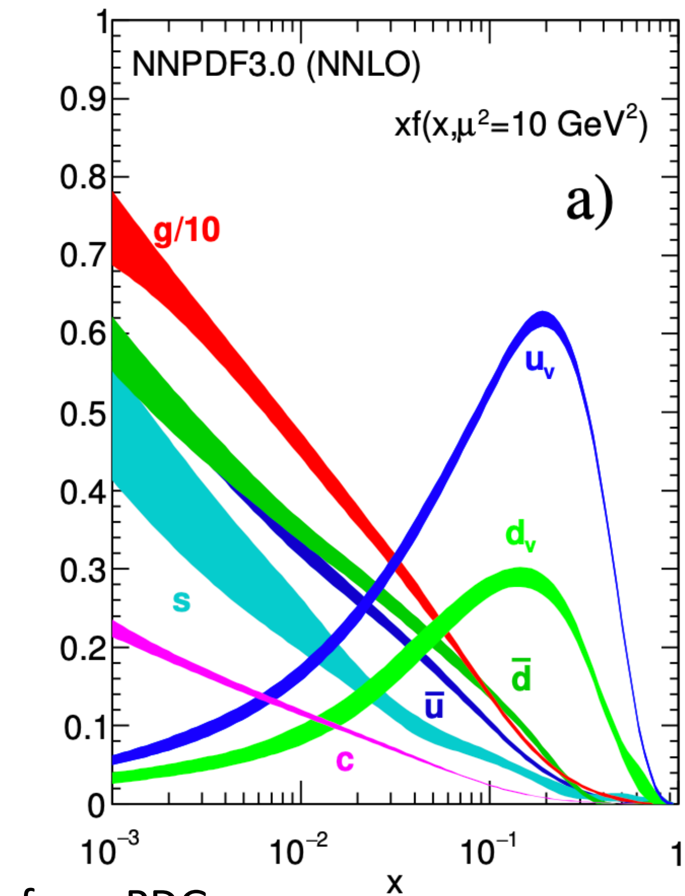
What are the data points we played with?

They represent a "property" of quarks in a proton..



proton at rest

proton with $v \simeq 1$
collection of «partons» (
valence and «sea» pa



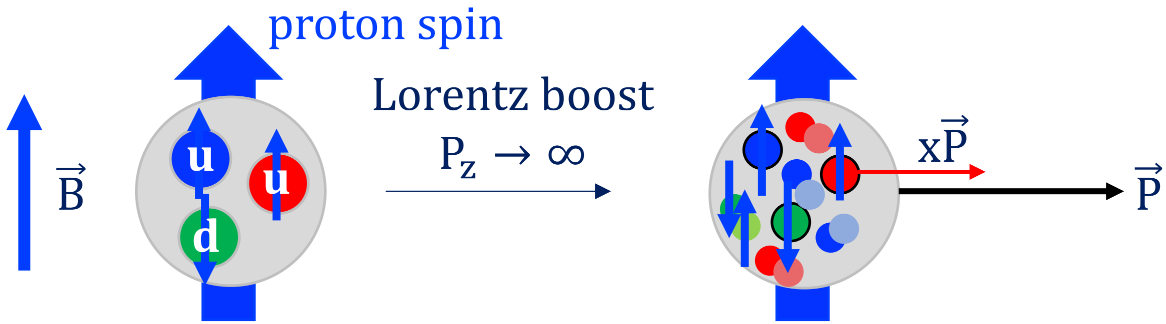
from PDG

Distribution of partons described by the
parton distribution function $\mathbf{f_1^q(x)}$

x is the fraction of the nucleon's momentum carried by quark (parton) q
(known as the «Bjorken variable»)

What are the data points we played with?

They represent a "property" of quarks in a proton.. the transverse polarization of up quarks



proton at rest

proton with $v \simeq 1$

collection of «partons» (quarks and gluons)

valence and «sea» partons

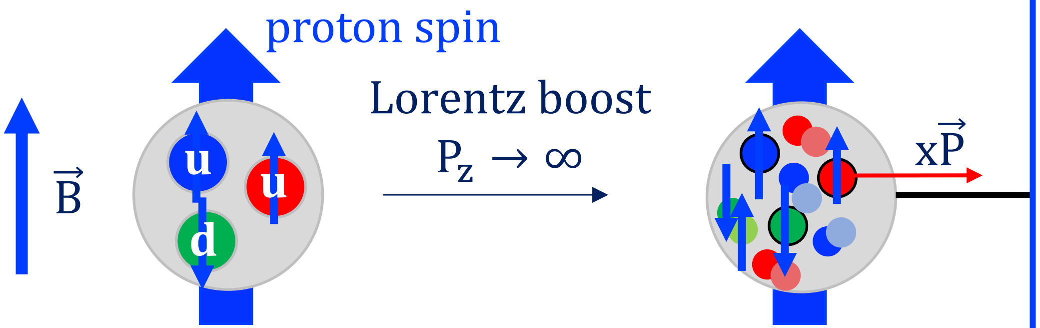
Feynman '68

Distribution of the transverse polarization of partons in a transversely polarized nucleon

→ Transversity PDF $\mathbf{h_1^q(x)}$

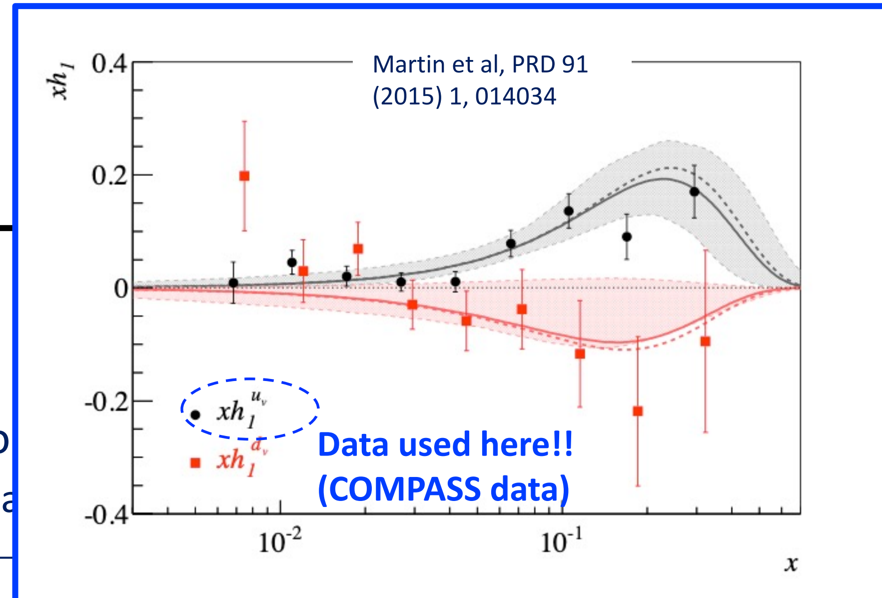
What are the data points we played with?

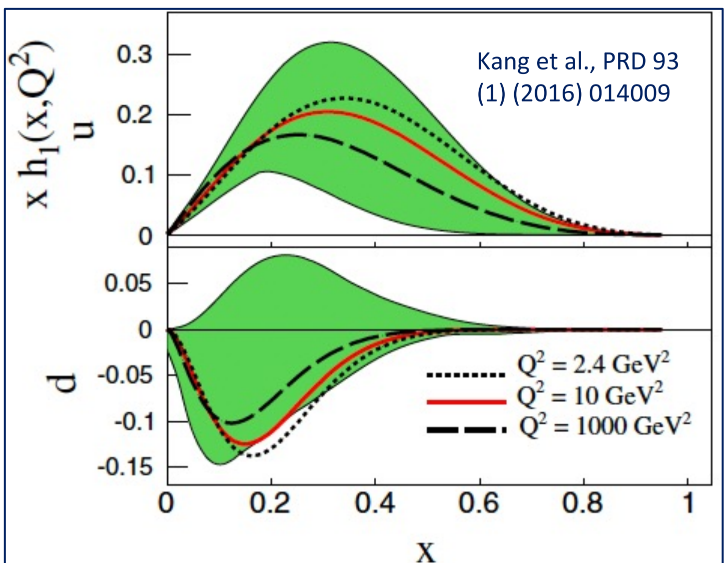
They represent a "property” of quarks in a proton.. the transverse polarization of up quarks

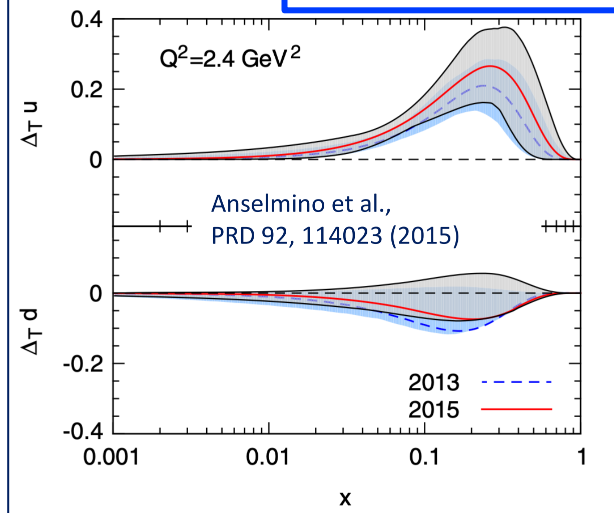


proton at rest

proton with v ≃ 1
collection of «parto
valence and «sea

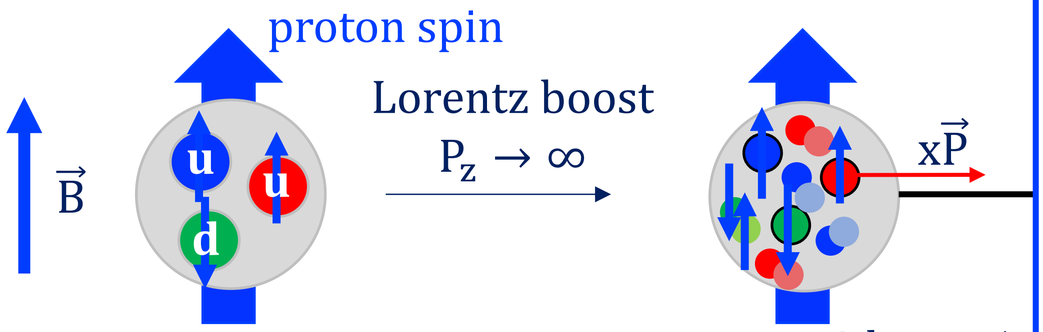






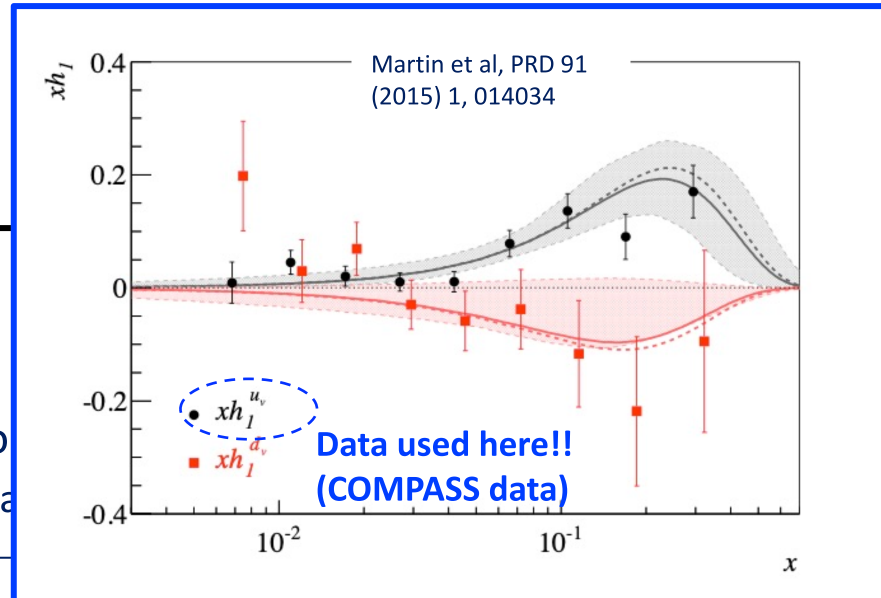
What are the data points we played with?

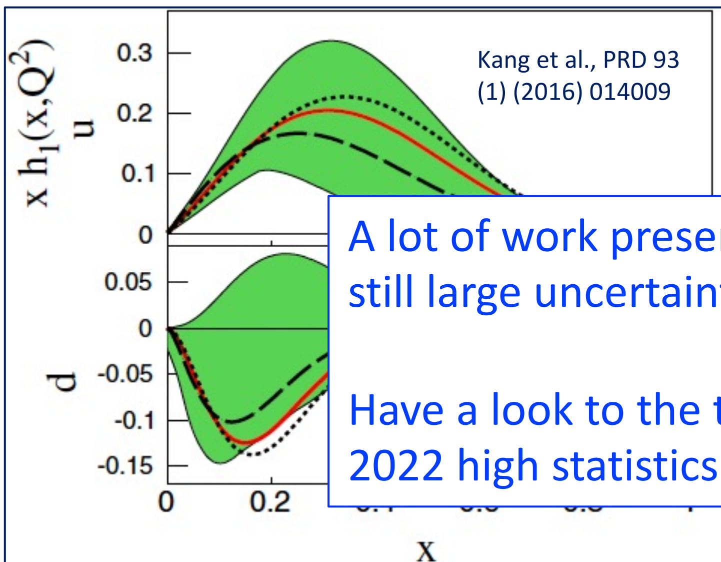
They represent a "property" of quarks in a proton.. the transverse polarization of up quarks

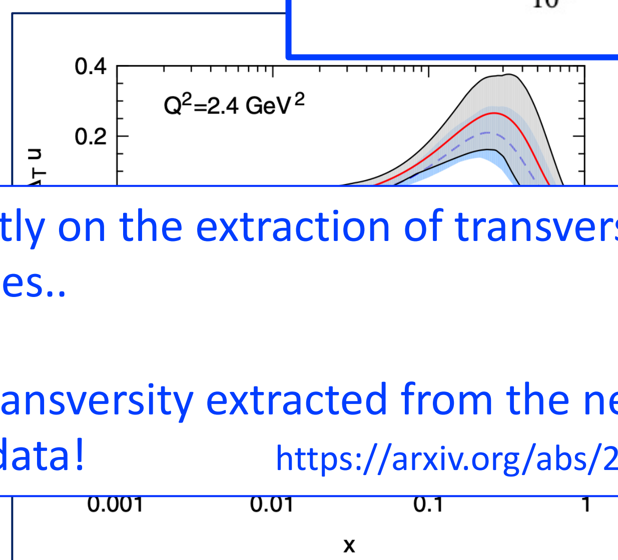


proton at rest

proton with v ≃ 1
collection of «parto
valence and «sea







A lot of work presently on the extraction of transversity
still large uncertainties..

Have a look to the transversity extracted from the new COMPASS 2022 high statistics data! https://arxiv.org/abs/2401.00309

Now that the steps for implementing the replica method are known, let's see the tools for the implementation

The ROOT framework

- "An open-source data analysis framework used by high energy physics and others.»
  quote taken from the website of the project «https://root.cern»
  born at CERN

- Heavily relies on the programming language C/C++
  object-oriented
  here assumed to be known [there is plenty of nice tutorials online..]

- Available for different platforms (Linux, MacOS, Windows)
  here assumed linux

- The first step, install ROOT following the instructions at https://root.cern/install/
  should already be installed in the lab's computers

- Run ROOT by the «root» command
  «root -l» if you'd like not to display the logo
  to quit root use unstead «.q»

ROOT interpreter

- After running «root», the ROOT interactive shell is opened

```
root [0] 1+2
(int) 3
root [1] pow(2,8)
(double) 256.00000
root [2] acos(-1.0)
(double) 3.1415927
root [3] TMath::Pi()
(double) 3.1415927
root [4]
```

ROOT has a command interpreter (CINT) for
C++ code   no ; needed at the end of a statement
(complex) mathematical operations which live in the TMath namespace

Functions in TMath begin with a capital letter, at variance with C++

```
root [0] double x = 0.1
(double) 0.10000000
root [1] TMath::Sin(x/x)
(double) 0.84147098
```

Evaluation of $f(x) = \frac{\sin x}{x}$ for $x = 0.1$

only C++ commands here

```
root [2] int N = 10
(int) 10
root [3] double sum = 0.0
(double) 0.0000000
root [4] for(int i=0; i<N; i++) sum += i
root [5] sum
(double) 45.000000
```

Sum of the first N=9 integers

$$S_N = \sum_{i=0}^{N} i = N(N+1)/2$$

$$S_9 = 9\times 10\times \frac{1}{2} = 45$$

- In general the analysis programs developed with ROOT are complex, with a lot of commmands → gathered in a «**macro**»

A simple macro

- A macro is essentially a function implementing the commands that constitute the analysis
- Here is an example of the implementation of the sum of first N integers in a macro

Users > albikerbizi > Desktop > ScuolaTirana > C ExampleMacro.C

```
1  // This is a simple ROOT macro.
2  #include "TROOT.h"    // ROOT header file.
3  #include <iostream>   // Standard input/output stream of C++
4  using namespace std;  // This allows to use commands like cout, endl, cin..
5  int main(){
6
7    int N = 10;
8    double sum = 0.0;
9    for(int i =0; i<N; i++) sum += i;
10   cout << "The result of the sum is " << sum << endl;
11   cout << "End of macro.\n";
12
13   return 0;
14 }
```

- Can be loaded in the ROOT interpreter by .L ExampleMacro.C
- And executed calling the function «main» main()

```
.L root [0] .L ExampleMacro.C
root [1] main()
The result of the sum is 45
End of macro.
(int) 0
root [2]
```

- Alternatively can be compiled as a C++ program using the comands
  g++ ExampleMacro.C -o ExampleMacro.o \`root-config --cflags --libs\`
  ./ExampleMacro.o

```
[albikerbizi@cli-10-106-4-193 ScuolaTirana % g++ ExampleMacro.C -o ExampleMacro.o `root-config --cflags --libs`
[albikerbizi@cli-10-106-4-193 ScuolaTirana % ./ExampleMacro.o
The result of the sum is 45
End of macro.
```

Note: When compiling the macro as a C++ program, treat it as a C++ code, e.g. include lines like 3 and 4 above..

Graphics: TCanvas class

- The TCanvas class allows to visualize graphics
- A TCanvas object can be declared and visualized by the following lines

```
// This is a simple ROOT macro.
#include "TROOT.h"    // ROOT header file.
#include "TCanvas.h"  // Header file with the TCanvas class.
#include <iostream>   // Standard input/output stream of C++
using namespace std;  // This allows to use commands like cout, endl, cin..

int main(){

  double width = 500;
  double height = 500;
  TCanvas *c = new TCanvas("c", "This is a canvas", width, height);
  c->SetLeftMargin(0.15);
  c->SetBottomMargin(0.15);
  c->cd();

  return 0;
}
```

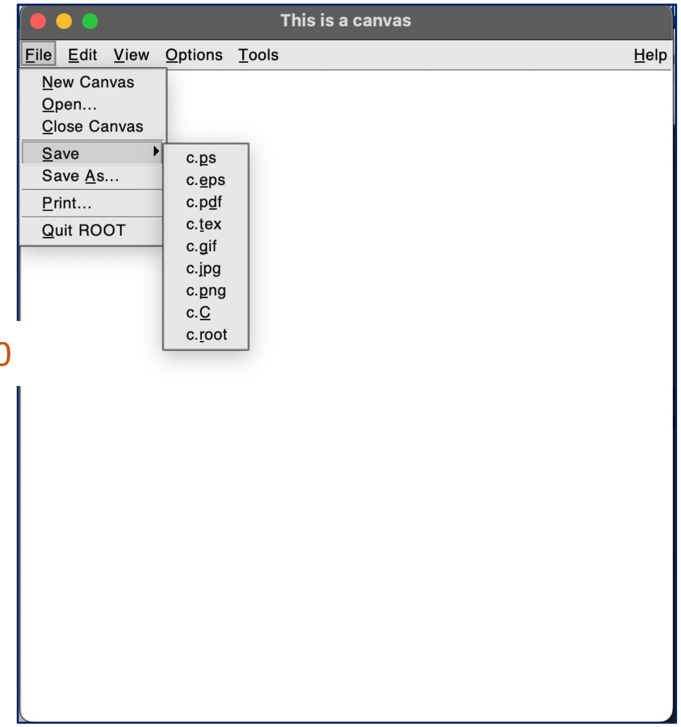
Name of the canvas

Titile of the canvas

Dimensions of the canvas

Empty space to the left and at the bottom of the canvas

cd in the canvas (to draw other objects)

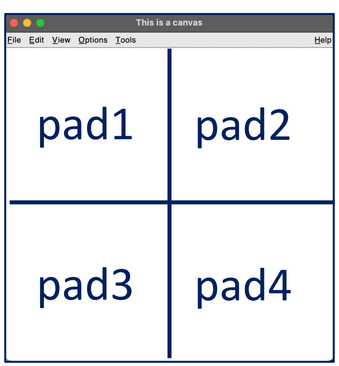


500

500

- The canvas can be divided by the lines

```
c->Divide(2,2);
c->cd(1);
//Draw something in pad 1.
//...
c->cd(4);
//.. Draw something in pad 4.
```



To save the canvas use

```
c->SaveAs("/path/to/directory/SavedCanvas.pdf");
```

or .png, .eps, jpg..

Histograms: TH1F and TH2F classes

- There are different classes that represent histograms, all inheriting from the class TH1
  for the complete guide see TH1 Class Reference

- Here we focus on 1D and 2D histograms with entries being floating points (prec. 7 digits)
  TH1F TH1F Class Reference
  TH2F TH2F Class Reference

Histograms: TH1F and TH2F classes

- There are different classes that represent histograms, all inheriting from the class TH1
  for the complete guide see [TH1 Class Reference]

- Here we focus on 1D and 2D histograms with entries being floating points (prec. 7 digits)
  TH1F [TH1F Class Reference]
  TH2F [TH2F Class Reference]

statistics box

- 1 D histograms

xmin xmax

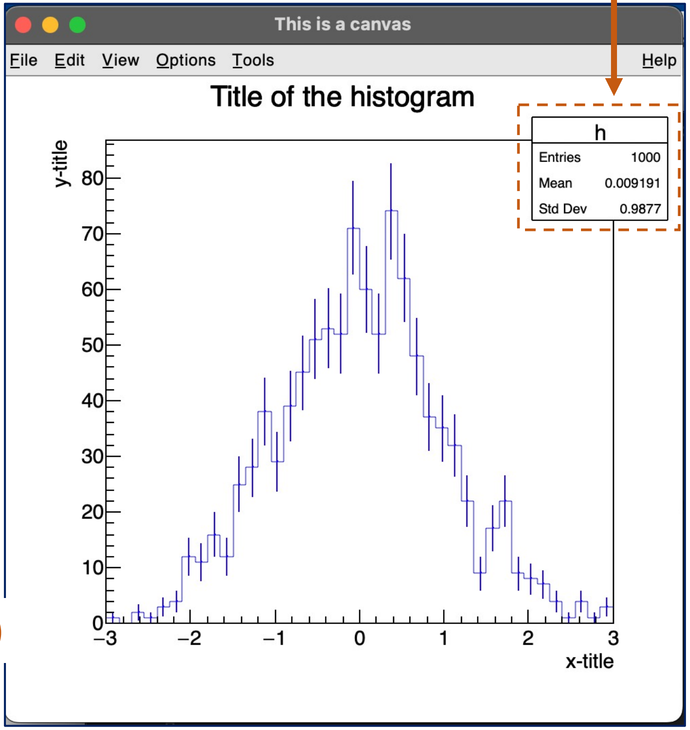
```
TH1F *h = new TH1F("h", "Title of the histogram", 40, -3, 3);
h->GetXaxis()->SetTitle("x-title");
h->GetYaxis()->SetTitle("y-title");
h->SetLineColor(kBlue);
h->SetLineColor(kBlue);
h->SetLineWidth(1);
h->SetLineStyle(1);
h->SetMarkerColor(kBlue);
//h->SetMarkerStyle(20);
h->FillRandom("gaus",1000);
h->Draw("HIST,E");
```

bins

cosmetics

Fill with a std gaussian

Draw «h» with boxes («HIST») and errorbars («E»)



To insert a single value x in the histogram
h->Fill(x)
Ex: within a loop to fill the histogram

Histograms: TH1F and TH2F classes

- There are different classes that represent histograms, all inheriting from the class TH1
  for the complete guide see [TH1 Class Reference]

- Here we focus on 1D and 2D histograms with entries being floating points (prec. 7 digits)
  TH1F [TH1F Class Reference]
  TH2F [TH2F Class Reference]

- 1 D histograms

xmin xmax

```
TH1F *h = new TH1F("h", "Title of the histogram", 40, -3, 3);
h->GetXaxis()->SetTitle("x-title");
h->GetYaxis()->SetTitle("y-title");
h->SetLineColor(kBlue);
h->SetLineColor(kBlue);
h->SetLineWidth(1);
h->SetLineStyle(1);
h->SetMarkerColor(kBlue);
//h->SetMarkerStyle(20);
h->FillRandom("gaus",1000);
h->Draw("HIST,E");
```

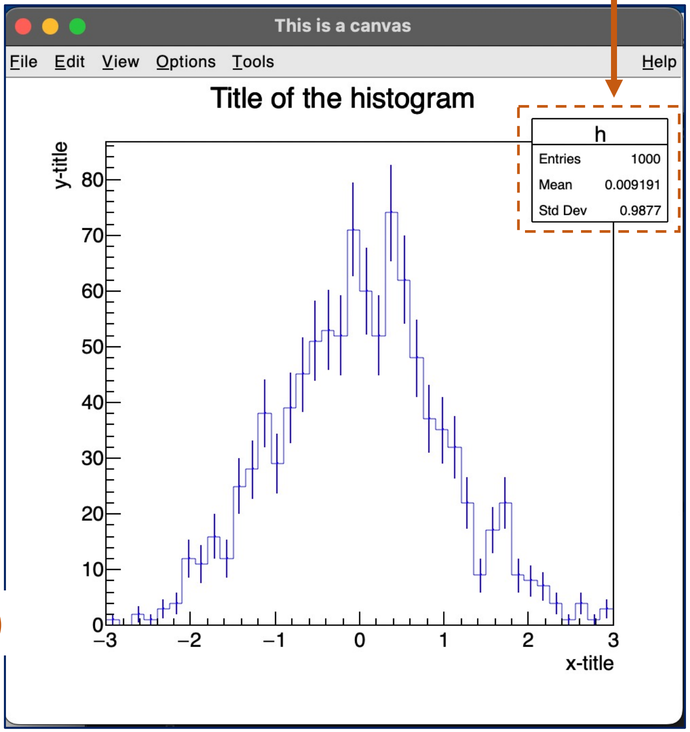
bins

cosmetics

Fill with a std gaussian

Draw «h» with boxes («HIST») and errorbars («E»)

statistics box

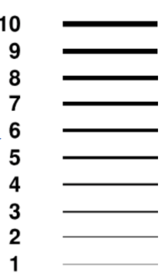


Color table, more in the [TColor Class Reference]

40	41	42	43	44	45	46	47	48	49
30	31	32	33	34	35	36	37	38	39
20	21	22	23	24	25	26	27	28	29
10	11	12	13	14	15	16	17	18	19
0	1	2	3	4	5	6	7	8	9

LineStyle

see [TAttLine]



LineWidth

Histograms: TH1F and TH2F classes

- There are different classes that represent histograms, all inheriting from the class TH1
  for the complete guide see [TH1 Class Reference]

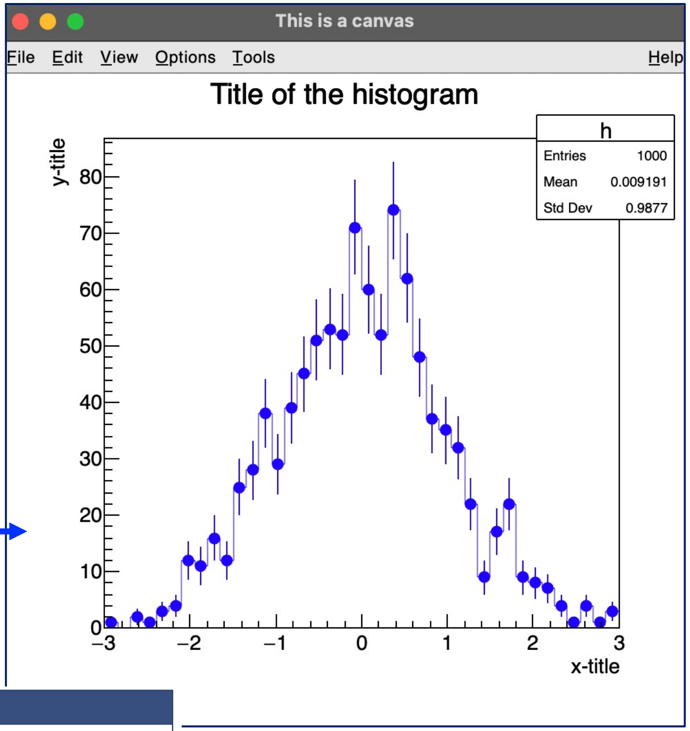
- Here we focus on 1D and 2D histograms with entries being floating points (prec. 7 digits)
  TH1F [TH1F Class Reference]
  TH2F [TH2F Class Reference]

- 1 D histograms

xmin xmax

```
TH1F *h = new TH1F("h", "Title of the histogram", 40, -3, 3);
h->GetXaxis()->SetTitle("x-title");
h->GetYaxis()->SetTitle("y-title");
h->SetLineColor(kBlue);
h->SetLineColor(kBlue);
h->SetLineWidth(1);
h->SetLineStyle(1);
h->SetMarkerColor(kBlue);
h->SetMarkerStyle(20); h->SetMarkerSize(0.8);
h->FillRandom("gaus",1000);
h->Draw("HIST,E");
```

bins



Draw options → see THistPaiter Class Reference

Option	Description
"E"	Draw error bars.
"AXIS"	Draw only axis.
"AXIG"	Draw only grid (if the grid is requested).
"HIST"	When an histogram has errors it is visualized by default with error bars. To visualize it without errors use the option "HIST" together with the required option (eg "hist same c"). The "HIST" option can also be used to plot only the histogram and not the associated function(s).
"FUNC"	When an histogram has a fitted function, this option allows to draw the fit result only.
"SAME"	Superimpose on previous picture in the same pad.
"SAMES"	Same as "SAME" and draw the statistics box

```
h->Draw("P,E");
```

for points with errorbars

Histograms: TH1F and TH2F classes

- There are different classes that represent histograms, all inheriting from the class TH1
  for the complete guide see [TH1 Class Reference]

- Here we focus on 1D and 2D histograms with entries being floating points (prec. 7 digits)
  TH1F [TH1F Class Reference]
  TH2F [TH2F Class Reference]

- 1 D histograms

Costumized binning can be provided in the declaration

```
const int Nbins = 6;
double xbins[Nbins+1] = {-3, -2, -1, 0, 1, 2, 3};
TH1F *h = new TH1F("h", "Title of the histogram", Nbins, xbins);
```

Other useful methods

```
for(int ibin=1; ibin<=h->GetNbinsX(); ibin++){
  double bin_content = h->GetBinContent(ibin);
  double bin_error = h->GetBinError(ibin);
  double bin_center = h->GetBinCenter(ibin);
  //..
  double counts = ..;
  h->SetBinContent(counts);
  h->SetBinError( sqrt(counts) );
}
```

For cycle running on all bins
Bin 0 underflow
Bin Nbins+1 overflow

```
TH1F * h2 = (TH1F*) h->Clone("h2");
h2->Divide(h);
h2->Add(h, 0.2);
```

h2 is a copy of h, with name «h2»
h2 is divided by h (division of bin contents, for each bin)
h is multiplied by 0.2 and added to h2 (bin-per-bin)

Histograms: TH1F and TH2F classes

- There are different classes that represent histograms, all inheriting from the class TH1
  for the complete guide see [TH1 Class Reference]

- Here we focus on 1D and 2D histograms with entries being floating points (prec. 7 digits)
  TH1F [TH1F Class Reference]
  TH2F [TH2F Class Reference]

- 1 D histograms
- **2D histograms**

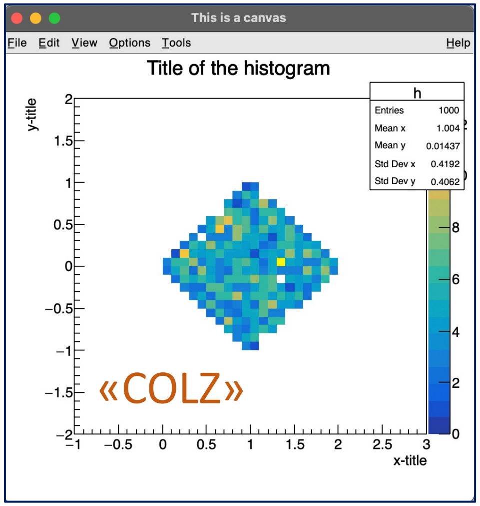
```
TRandom2 *rndGen = new TRandom2();
TH2F * h = new TH2F("h","Title of the histogram", 40, -1.0, 3.0, 40, -2.0, 2.0);
// Same methods as TH1F to customize the histogram.
h->GetXaxis()->SetTitle("x-title");
h->GetYaxis()->SetTitle("y-title");
//...
for(int i=0; i<1000; i++){
  double x = rndGen->Uniform(0.0, 1.0);
  double y = rndGen->Uniform(0.0, 1.0);
 h->Fill(x+y, x-y);
}
h->Draw("COLZ");
h->Draw("LEGO");
```

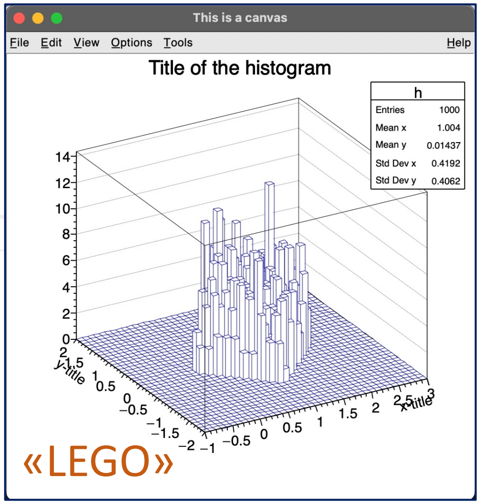
bins X   xmin   # bins Y   ymax

xmax   ymin

Filling
here using a random number generator, provided by the class TRandom2

Draw options





Graphs: TGraph and TGraphErrors classes

- The TGraph and TGraphErrors classes allow to construct graphs with
  - only points TGraph
  - points and errors TGraphError

- The constructors are similar
  TGraph *gr1 = new TGraph(int nPts, double xPts, double yPts);
  TGraphErrors *gr2 = new TGraphErrors(int nPts, double xPts, double yPts, double xErr, double yErr);

  nPts = number of points
  xPts = array of dimension nPts with x-values
  yPts = array of dimension nPts with y-values
  xErr / yErr = array of dimension nPts with uncertainties on x / y

Graphs: TGraph and TGraphErrors classes

- The TGraph and TGraphErrors classes allow to construct graphs with
  only points TGraph
  points and errors TGraphError

- The constructors are similar
  TGraph *gr1 = new TGraph(int nPts, double xPts, double yPts);
  TGraphErrors *gr2 = new TGraphErrors(int nPts, double xPts, double yPts, double xErr, double yErr);

nPts = number of points
xPts = array of dimension nPts with x-values
yPts = array of dimension nPts with y-values
xErr / yErr = array of dimension nPts with uncertainties on x / y

```
#include "TGraph.h"   // Header with TGraph class
#include "TGraphErrors.h" // Header with TGraphErrors class.

int main(){

  const int nPts = 10;
  double xPts[nPts], yPts[nPts];
  double xErr[nPts], yErr[nPts];
  double xPts_2[nPts];
  TRandom2 *rnd = new TRandom2();
  for(int i=0; i<nPts; i++){
    xPts[i] = 0.1+i*(1.0-0.1)/nPts; // Example of x-points
    yErr[i] = 0.04; xErr[i] = 0.03; // Uncertainties on y and x.
    yPts[i] = 1.0*xPts[i]+rnd->Gaus()*yErr[i];  // Example of y points.
    xPts_2[i] = xPts[i]+0.02; // Small shift for the second graph.
  }

  TGraph *gr1 = new TGraph(nPts, xPts, yPts);
  gr1->SetMarkerColor(kBlue);
  gr1->SetMarkerStyle(20);
  gr1->SetMarkerSize(0.8);

  TGraphErrors *gr2 = new TGraphErrors(nPts, xPts_2, yPts, xErr, yErr);
  gr2->SetMarkerColor(kRed);
  gr2->SetMarkerStyle(24);
  gr2->SetMarkerSize(0.8);
```

cosmetics

cosmetics

Graphs: TGraph and TGraphErrors classes

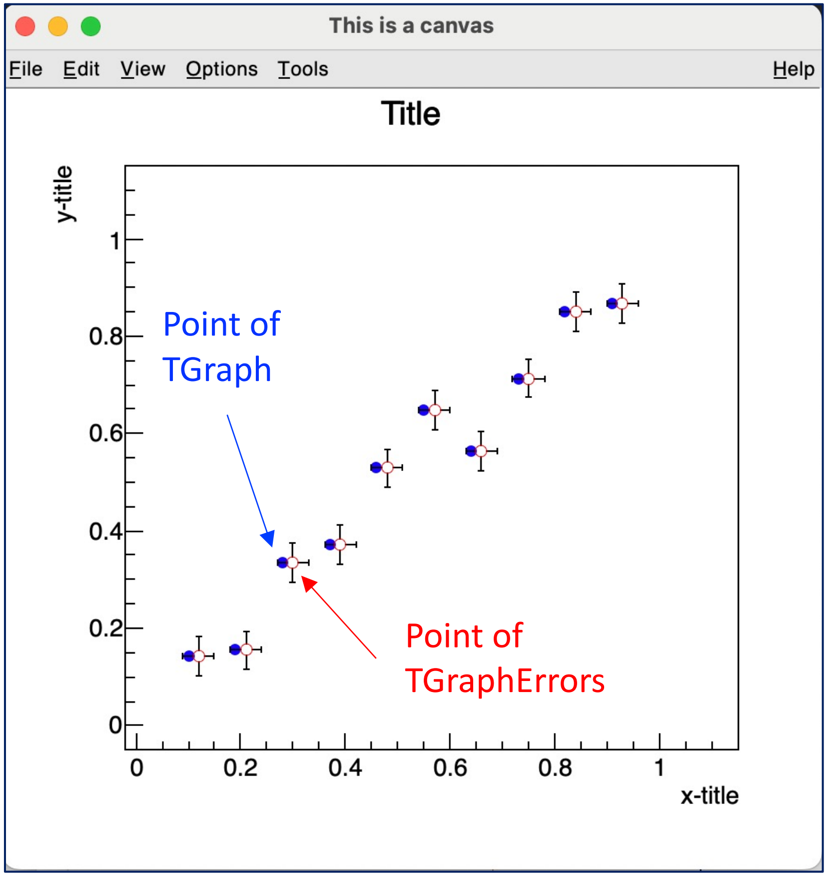
- The TGraph and TGraphErrors classes allow to construct graphs with
  - only points — TGraph
  - points and errors — TGraphError

- The constructors are similar
  - TGraph *gr1 = new TGraph(int nPts, double xPts, double yPts);
  - TGraphErrors *gr2 = new TGraphErrors(int nPts, double xPts, double yPts, double xErr, double yErr);

```
#include "TGraph.h"   // Header with TGraph class
#include "TGraphErrors.h" // Header with TGraphErrors class.

int main(){

  const int nPts = 10;
  double xPts[nPts], yPts[nPts];
  double xErr[nPts], yErr[nPts];
  double xPts_2[nPts];
  TRandom2 *rnd = new TRandom2();
  for(int i=0; i<nPts; i++){
    xPts[i] = 0.1+i*(1.0-0.1)/nPts; // Example of x-points
    yErr[i] = 0.04; xErr[i] = 0.03; // Uncertainties on y and x.
    yPts[i] = 1.0*xPts[i]+rnd->Gaus()*yErr[i];   // Example of y points.
    xPts_2[i] = xPts[i]+0.02; // Small shift for the second graph.
  }

  TGraph *gr1 = new TGraph(nPts, xPts, yPts);
  gr1->SetMarkerColor(kBlue);
  gr1->SetMarkerStyle(20);
  gr1->SetMarkerSize(0.8);

  TGraphErrors *gr2 = new TGraphErrors(nPts, xPts_2, yPts, xErr, yErr);
  gr2->SetMarkerColor(kRed);
  gr2->SetMarkerStyle(24);
  gr2->SetMarkerSize(0.8);
```

cosmetics

cosmetics

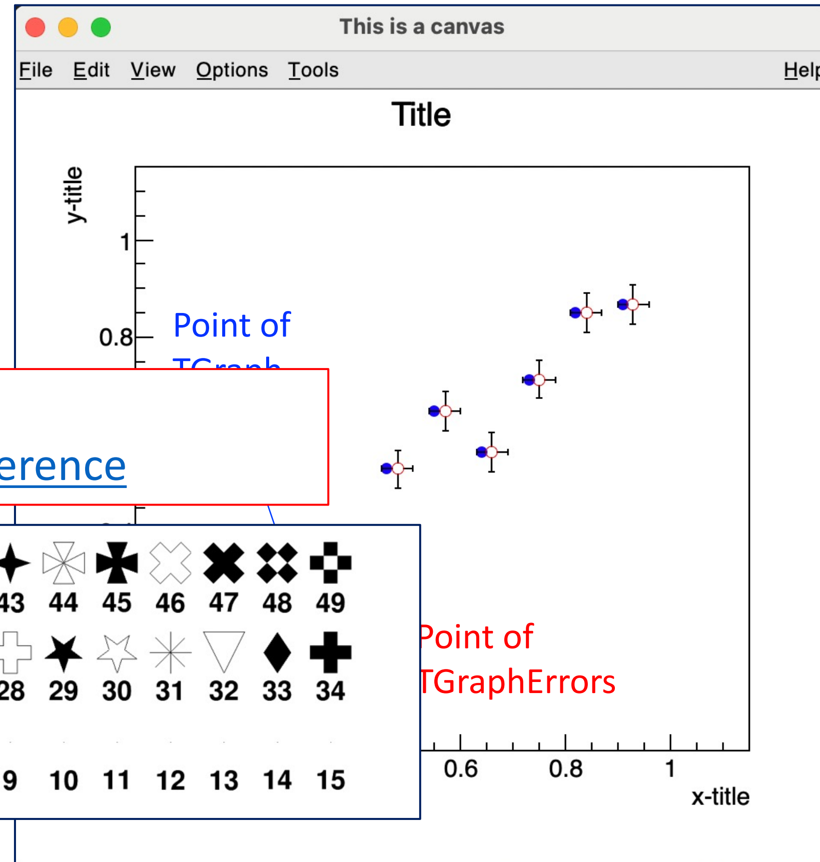


Graphs: TGraph and TGraphErrors classes

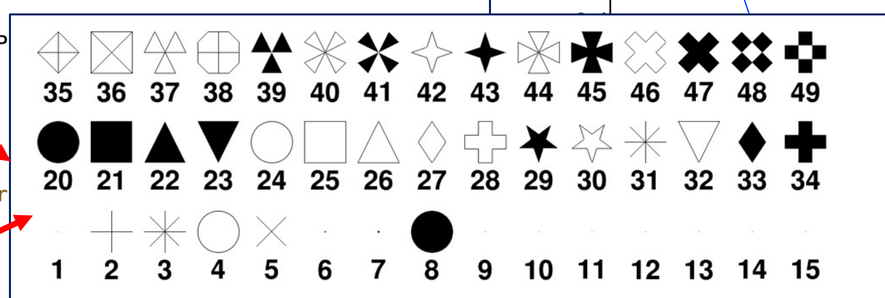
- The TGraph and TGraphErrors classes allow to construct graphs with
  - only points — TGraph
  - points and errors — TGraphError

- The constructors are similar
  - TGraph *gr1 = new TGraph(int nPts, double xPts, double yPts);
  - TGraphErrors *gr2 = new TGraphErrors(int nPts, double xPts, double yPts, double xErr, double yErr);

```
#include "TGraph.h"   // Header with TGraph class
#include "TGraphErrors.h" // Header with TGraphErrors class.

int main(){

  const int nPts = 10;
  double xPts[nPts], yPts[nPts];
  double xErr[nPts], yErr[nPts];
  double xPts_2[nPts];
  TRandom2 *rnd = new TRandom2();
  for(int i=0; i<nPts; i++){
    xPts[i] = 0.1+i*(1.0-0.1)/nP
    yErr[i] = 0.04; xErr[i] = 0.
    yPts[i] = 1.0*xPts[i]+rnd->G
    xPts_2[i] = xPts[i]+0.02; //
  }

  TGraph *gr1 = new TGraph(nPts, xP
  gr1->SetMarkerColor(kBlue);
  gr1->SetMarkerStyle(20);
  gr1->SetMarkerSize(0.8);

  TGraphErrors *gr2 = new TGraphErr
  gr2->SetMarkerColor(kRed);
  gr2->SetMarkerStyle(24);
  gr2->SetMarkerSize(0.8);
```



Many marker styles,
see [TAttMarker Class Reference]



Functions: TF1 class

- The TF1 class allows to represent functions

- The general way of defining a TF1 object is the following

pointer to x-values

pointer to parameter values

```
#include "TF1.h"

double myFunction(double *x, double*par){
  double p0 = par[0];
  double p1 = par[1];
  double X = x[0];
  return p0 + p1 * X;
}
```

C++ function

value of the function, given x and par

```
int main(){

  TF1 *f = new TF1("f", myFunction, 0.0, 1.0, 2);
  f->SetParName(0,"q");
  f->SetParName(1,"m");
  f->SetParameter(0,-0.05);
  f->SetParameter(1, 0.8);
  f->SetLineStyle(1);
  f->SetLineWidth(2);
  f->SetLineColor(kRed);
```

name

C++ func.

xmin xmax

number of parameters

names and values of parameters

cosmetics

Functions: TF1 class

- The TF1 class allows to represent functions

- The general way of defining a TF1 object is the following

pointer to x-values

pointer to parameter values

```
#include "TF1.h"

double myFunction(double *x, double*par){
  double p0 = par[0];
  double p1 = par[1];
  double X = x[0];
  return p0 + p1 * X;
}
```

C++ function

value of the function, given x and par

name

C++ func.

number of parameters

```
int main(){

  TF1 *f = new TF1("f", myFunction, 0.0, 1.0, 2);
  f->SetParName(0,"q");
  f->SetParName(1,"m");
  f->SetParameter(0,-0.05);
  f->SetParameter(1, 0.8);
  f->SetLineStyle(1);
  f->SetLineWidth(2);
  f->SetLineColor(kRed);
```

xmin xmax

names and values of parameters

cosmetics

How to draw the function?

Functions: TF1 class

- The TF1 class allows to represent functions
- The general way of defining a TF1 object is the following

pointer to x-values

pointer to parameter values

```
#include "TF1.h"

double myFunction(double *x, double*par){
  double p0 = par[0];
  double p1 = par[1];
  double X = x[0];
  return p0 + p1 * X;
}
```

C++ function

value of the function, given x and par

```
int main(){

  TF1 *f = new TF1("f", myFunction, 0.0, 1.0, 2);
  f->SetParName(0,"q");
  f->SetParName(1,"m");
  f->SetParameter(0,-0.05);
  f->SetParameter(1, 0.8);
  f->SetLineStyle(1);
  f->SetLineWidth(2);
  f->SetLineColor(kRed);
```

name C++ func. xmin xmax number of parameters

names and values of parameters

cosmetics

How to draw the function?

First a TH2F for the axes

Then the function

```
TCanvas *c = new TCanvas("c", "This is a canvas", 500, 500);
c->SetLeftMargin(0.15); c->SetBottomMargin(0.15); c->cd();
TH2F *h = new TH2F("h", "Title", 10,-0.02,1.15,10,-0.05,1.15);
h->GetXaxis()->SetTitle("x-title");
h->GetYaxis()->SetTitle("y-title");
h->SetStats(kFALSE);
h->Draw("");
f->Draw("l,R,same");

return 0;
}
```

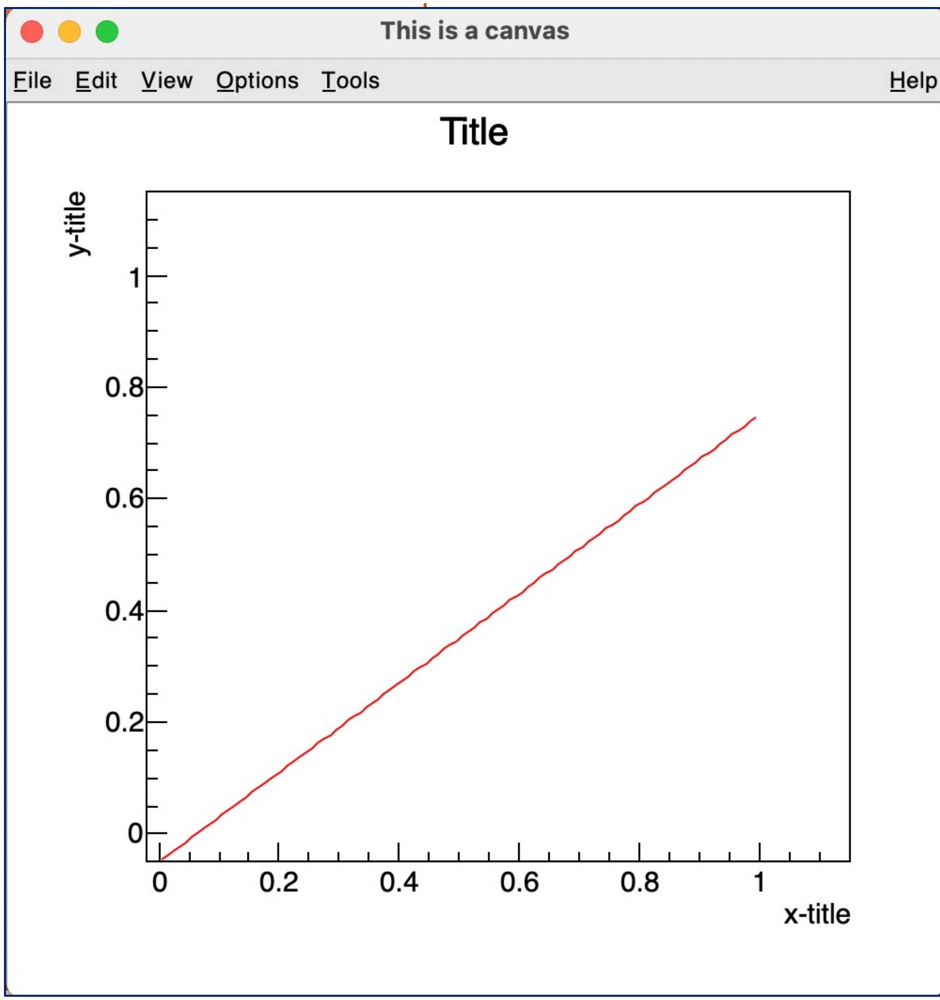
Draw with lines (l), in the range xmin-xmax (R) on top of the histogram h (same)

Functions: TF1 class

- The TF1 class allows to represent functions
- The general way of defining a TF1 object is the following

pointer to

pointer to parameter values



on

ar

How to draw the function?
First a TH2F for the axes
Then the function

```
TCanvas *c = new TCanvas("c", "This is a canvas", 500, 500);
c->SetLeftMargin(0.15); c->SetBottomMargin(0.15); c->cd();
TH2F *h = new TH2F("h", "Title", 10,-0.02,1.15,10,-0.05,1.15);
h->GetXaxis()->SetTitle("x-title");
h->GetYaxis()->SetTitle("y-title");
h->SetStats(kFALSE);
h->Draw("");
f->Draw("l,R,same");

return 0;
}
```

Draw with lines (l), in the range xmin-xmax (R) on top of the histogram h (same)

More info in the [TF1 Class Reference]

Fits to histograms and graphs

- ROOT offers an automatic way of fitting histograms (here 1D) and graphs
- The objects that we need are

TH1F or TGraphErrors	already seen
TF1	already seen
rules for the fitting	this slide

- Fit to a TH1F
  example, a histogram filled with random numbers generated according to a standard gaussian distrib.

Fits to histograms and graphs

- ROOT offers an automatic way of fitting histograms (here 1D) and graphs
- The objects that we need are

TH1F or TGraphErrors	already seen
TF1	already seen
rules for the fitting	this slide

- Fit to a TH1F
  example, a histogram filled with random numbers generated according to a standard gaussian distrib.

```
double Gaussian(double *x, double*par){
  double N = par[0];
  double mu = par[1];
  double sigma = par[2];
  double X = x[0];
  return N * TMath::Exp(-TMath::Power((X-mu)/sigma,2)/2.0);
}

int main(){
  TH1F *h = new TH1F("h","Gaussian distribution",40,-5.0,5.0);
  h->GetXaxis()->SetTitle("x");
  h->GetYaxis()->SetTitle("counts");
  h->FillRandom("gaus");

  TF1 *f = new TF1("f", Gaussian, -3.0, 3.0, 3);
  f->SetParNames("N","mu","sigma");
  f->SetParameters(100,0.0,1.0);
  f->SetLineColor(kBlue);
  h->Fit(f,"R0");
```

Definition of the gaussian, returns

$$N\times \exp[-\frac{(x-\mu)^2}{2\sigma^2}]$$

N normalization
μ mean value
σ standard deviation

Histogram to be fitted

Function to be used for the fitting

Fit operation
"R" fit in the range of f
"0" do not draw f automatically each time h is drawn
Default fit procedure → χ^2 **minimization**

Fits to histograms and graphs

- ROOT offers an automatic way of fitting histograms (here 1D) and graphs
- The objects that we need are

TH1F or TGraphErrors	already seen
TF1	already seen
rules for the fitting	this slide

- Fit to a TH1F

  example, a histogram filled with random numbers generated according to a standard gaussian distrib.

```
double Gaussian(double *x, double*par){
  double N = par[0];
  double mu = par[1];
  double sigma = par[2];
  double X = x[0];
  return N * TMath::Exp(-TMath::Power(
}

int main(){
  TH1F *h = new TH1F("h","Gaussian dist
  h->GetXaxis()->SetTitle("x");
  h->GetYaxis()->SetTitle("counts");
  h->FillRandom("gaus");

  TF1 *f = new TF1("f", Gaussian, -3.0,
  f->SetParNames("N","mu","sigma");
  f->SetParameters(100,0.0,1.0);
  f->SetLineColor(kBlue);
  h->Fit(f,"R0");
```

Fit operation, with options

"R" fit in the range of f

"0" do not draw f automatically each time h is drawn

Default fit procedure → χ^2 **minimization**

Some of the fit options, see description of TH1F:Fit() method in TH1F Class Reference, for more

option	description
"L"	Uses a log likelihood method (default is chi-square method). To be used when the histogram represents counts.
"WL"	Weighted log likelihood method. To be used when the histogram has been filled with weights different than 1. This is needed for getting correct parameter uncertainties for weighted fits.
"P"	Uses Pearson chi-square method. Uses expected errors instead of the observed one (default case). The expected error is instead estimated from the square-root of the bin function value.
"MULTI"	Uses Loglikelihood method based on multi-nomial distribution. In this case the function must be normalized and one fits only the function shape.
"W"	Fit using the chi-square method and ignoring the bin uncertainties and skip empty bins.
"WW"	Fit using the chi-square method and ignoring the bin uncertainties and include the empty bins.
"I"	Uses the integral of function in the bin instead of the default bin center value.
"F"	Uses the default minimizer (e.g. Minuit) when fitting a linear function (e.g. polN) instead of the linear fitter.
"U"	Uses a user specified objective function (e.g. user providedlikelihood function) defined using `TVirtualFitter::SetFCN`
"E"	Performs a better parameter errors estimation using the Minos technique for all fit parameters.
"M"	Uses the IMPROVE algorithm (available only in TMinuit). This algorithm attempts improve the found local minimum by searching for a better one.
"S"	The full result of the fit is returned in the `TFitResultPtr`. This is needed to get the covariance matrix of the fit. See `TFitResult` and the base class `ROOT::Math::FitResult`.
"Q"	Quiet mode (minimum printing)

Fits to histograms and graphs

- ROOT offers an automatic way of fitting histograms (here 1D) and graphs
- The objects that we need are

TH1F or TGraphErrors	already seen
TF1	already seen
rules for the fitting	this slide

- Fit to a TH1F

  example, a histogram filled with random numbers generated according to a standard gaussian distrib.

```
double Gaussian(double *x, double*par){
  double N = par[0];
  double mu = par[1];
  double sigma = par[2];
  double X = x[0];
  return N * TMath::Exp(-TMath::Power((X-mu)/sigma,2)/2.0);
}

int main(){
  TH1F *h = new TH1F("h","Gaussian distribution",40,-5.0,5.0);
  h->GetXaxis()->SetTitle("x");
  h->GetYaxis()->SetTitle("counts");
  h->FillRandom("gaus");

  TF1 *f = new TF1("f", Gaussian, -3.0, 3.0, 3);
  f->SetParNames("N","mu","sigma");
  f->SetParameters(100,0.0,1.0);
  f->SetLineColor(kBlue);
  h->Fit(f,"R0");

  TCanvas *c = new TCanvas("c", "This is a canvas", 500, 500);
  c->SetLeftMargin(0.15); c->SetBottomMargin(0.15); c->cd();
  gStyle->SetOptFit(1111);
  h->Draw("HIST,E");
  f->Draw("l,R,same");

  return 0;
}
```

To show the fit results in the stat box of the histogram

Draw the histogram and function

Fits to histograms and graphs

- ROOT offers an automatic way of fitting histograms (here 1D) and graphs
- The objects that we need are

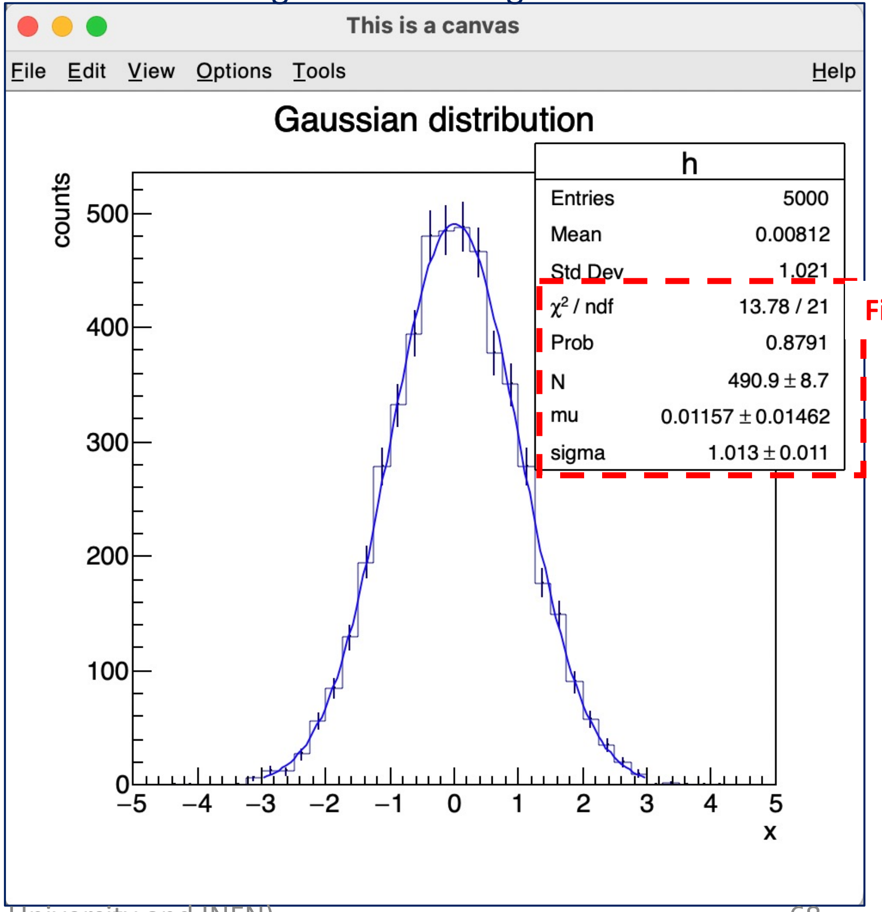
TH1F or TGraphErrors	already seen
TF1	already seen
rules for the fitting	this slide

- Fit to a TH1F

  example, a histogram filled with random numbers generated according to a standard gaussian distrib.

```
double Gaussian(double *x, double*par){
  double N = par[0];
  double mu = par[1];
  double sigma = par[2];
  double X = x[0];
  return N * TMath::Exp(-TMath::Power((X-mu)/sigma,2)/2.0);
}

int main(){
  TH1F *h = new TH1F("h","Gaussian distribution",40,-5.0,5.0);
  h->GetXaxis()->SetTitle("x");
  h->GetYaxis()->SetTitle("counts");
  h->FillRandom("gaus");

  TF1 *f = new TF1("f", Gaussian, -3.0, 3.0, 3);
  f->SetParNames("N","mu","sigma");
  f->SetParameters(100,0.0,1.0);
  f->SetLineColor(kBlue);
  h->Fit(f,"R0");

  TCanvas *c = new TCanvas("c", "This is a canvas", 500, 500);
  c->SetLeftMargin(0.15); c->SetBottomMargin(0.15); c->cd();
  gStyle->SetOptFit(1111);
  h->Draw("HIST,E");
  f->Draw("l,R,same");

  return 0;
}
```

To show the fit results in the stat box of the histogram



Fit info

Fits to histograms and graphs

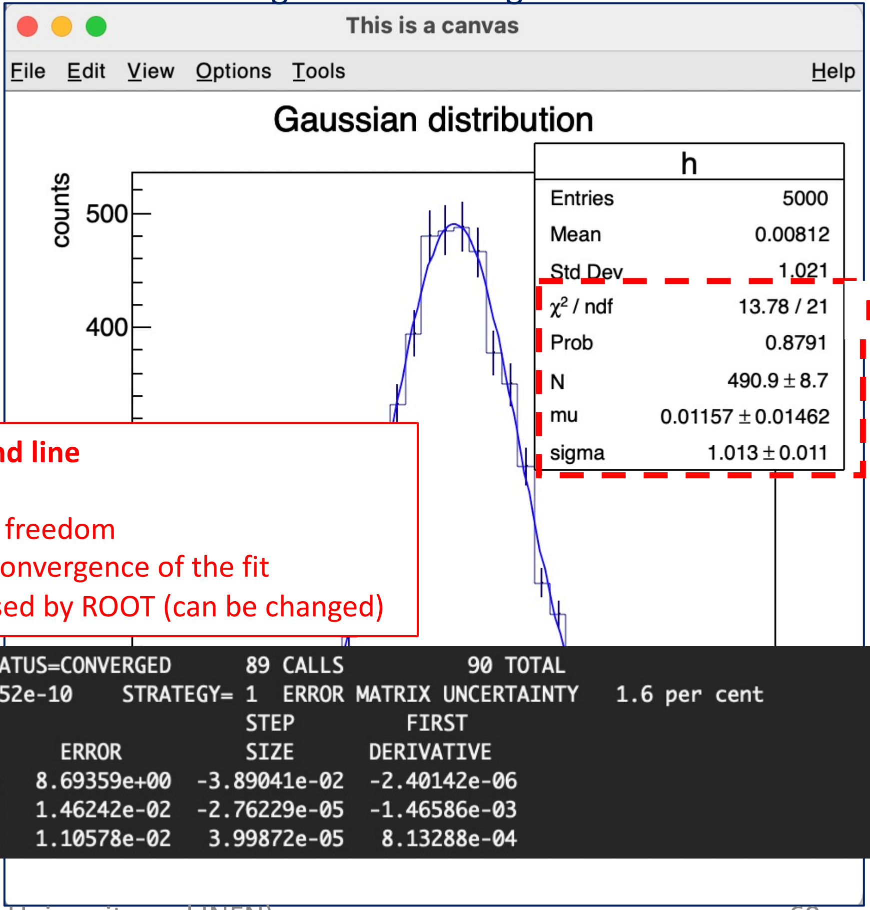
- ROOT offers an automatic way of fitting histograms (here 1D) and graphs
- The objects that we need are

TH1F or TGraphErrors	already seen
TF1	already seen
rules for the fitting	this slide

- Fit to a TH1F
  example, a histogram filled with random numbers generated according to a standard gaussian distrib.

```
double Gaussian(double *x, double*par){
  double N = par[0];
  double mu = par[1];
  double sigma = par[2];
  double X = x[0];
  return N * TMath::Exp(-TMath::Power((X-mu)/sigma,2)/2.0);
}

int main(){
  TH1F *h = new TH1F("h","Gaussian distribution",40,-5.0,5.0);
  h->GetXaxis()->SetTitle("x");
  h->GetYaxis()->SetTitle("counts");
  h->FillRandom("gaus");

  TF1 *f = new TF1("f", Gaussian,
  f->SetParNames("N","mu","sigma")
  f->SetParameters(100,0.0,1.0);
  f->SetLineColor(kBlue);
  h->Fit(f,"R0");

  TCanvas *c = new TCanvas("c", "T
  c->SetLeftMargin(0.15); c->SetB
  gStyle->SetOptFit(1111);
  h->Draw("HIST,E");
  f->Draw("l,R,same");

  return 0;
}
```

To sh
stat



Fit info

Fit info also in the command line

FCN	χ^2
Ndf	# degrees of freedom
STATUS	info on the convergence of the fit
MIGRAD	minimizer used by ROOT (can be changed)

```
FCN=13.7753 FROM MIGRAD    STATUS=CONVERGED      89 CALLS          90 TOTAL
                     EDM=3.14652e-10    STRATEGY= 1  ERROR MATRIX UNCERTAINTY   1.6 per cent
 EXT PARAMETER                                   STEP         FIRST
 NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
  1  N            4.90912e+02   8.69359e+00  -3.89041e-02  -2.40142e-06
  2  mu           1.15694e-02   1.46242e-02  -2.76229e-05  -1.46586e-03
  3  sigma        1.01300e+00   1.10578e-02   3.99872e-05   8.13288e-04
```

Fits to histograms and graphs

- ROOT offers an automatic way of fitting histograms (here 1D) and graphs
- The objects that we need are

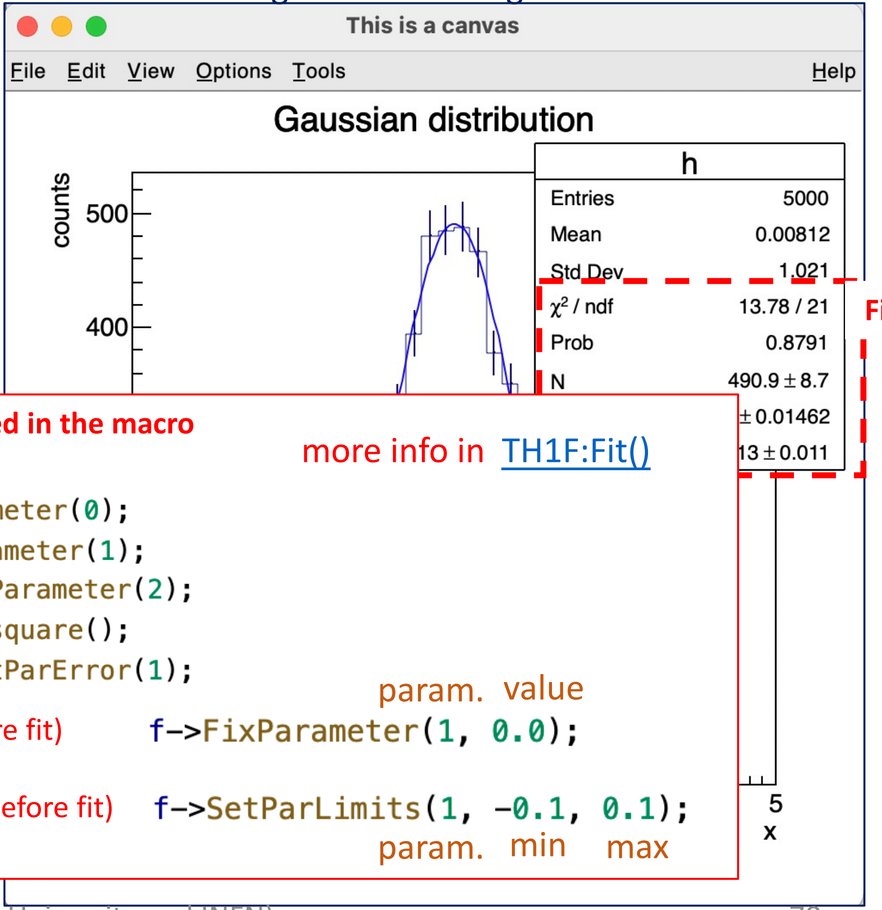
TH1F or TGraphErrors	already seen
TF1	already seen
rules for the fitting	this slide

- Fit to a TH1F

  example, a histogram filled with random numbers generated according to a standard gaussian distrib.

```
double Gaussian(double *x, double*par){
  double N = par[0];
  double mu = par[1];
  double sigma = par[2];
  double X = x[0];
  return N * TMath::Exp(-TMath::Power((X-mu)/sigma,2)/2.0);
}

int main(){
  TH1F *h = new TH1F("h","Gaussian distribution",40,-5.0,5.0);
  h->GetXaxis()->SetTitle("x");
  h->GetYaxis()->SetTitle("counts");
  h->FillRandom("gaus");

  TF1 *f = new TF1("f", Gaussian,
  f->SetParNames("N","mu","sigma")
  f->SetParameters(100,0.0,1.0);
  f->SetLineColor(kBlue);
  h->Fit(f,"R0");

  TCanvas *c = new TCanvas("c", "T
  c->SetLeftMargin(0.15); c->SetBo
  gStyle->SetOptFit(1111);
  h->Draw("HIST,E");
  f->Draw("l,R,same");

  return 0;
}
```

To sh
stat b



Fit info

Fit info can also be accessed in the macro

more info in [TH1F:Fit()]

```
double N = f->GetParameter(0);
double mu = f->GetParameter(1);
double sigma = f->GetParameter(2);
double X2 = f->GetChisquare();
double err_mu = f->GetParError(1);
```

Fix parameter values (before fit) `f->FixParameter(1, 0.0);` (param. value)

Set limits for parameters (before fit) `f->SetParLimits(1, -0.1, 0.1);` (param. min max)

Fits to histograms and graphs

- ROOT offers an automatic way of fitting histograms (here 1D) and graphs
- The objects that we need are

  TH1F or TGraphErrors already seen
  TF1 already seen
  rules for the fitting this slide
- Fit to a TH1F
- Fit to a TGraphErrors

  as for TH1F, using the gr->Fit(f,"opt") gr being the pointer to a TGraphErrors
  f pointer to the fit function
  opt options, e.g. opt = R0

default minimizing techinque X2, can be changed as for TH1F

Fits to histograms and graphs

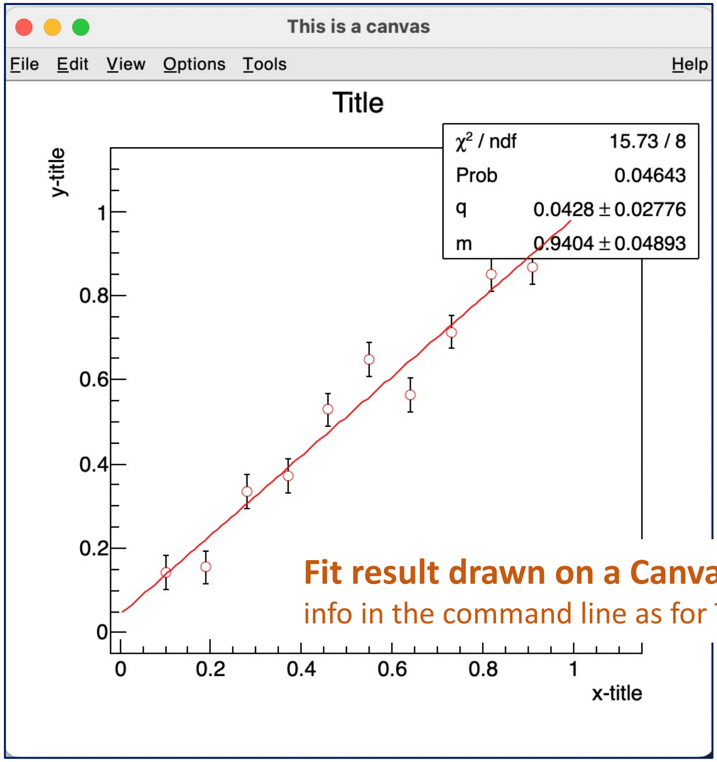
- ROOT offers an automatic way of fitting histograms (here 1D) and graphs
- The objects that we need are

TH1F or TGraphErrors	already seen
TF1	already seen
rules for the fitting	this slide

- Fit to a TH1F
- Fit to a TGraphErrors
  - as for TH1F, using the gr->Fit(f,"opt")

```cpp

double myFunction(double *x, double*par){
  return par[0] + par[1] * x[0];
}

int main(){

  const int nPts = 10;
  double xPts[nPts], yPts[nPts], yErr[nPts];
  TRandom2 *rnd = new TRandom2();
  for(int i=0; i<nPts; i++){
    xPts[i] = 0.1+i*(1.0-0.1)/nPts; // Example of x-points
    yErr[i] = 0.04;                 // Uncertainties on y.
    yPts[i] = 1.0*xPts[i]+rnd->Gaus()*yErr[i];  // Example of y points.
  }

  TGraphErrors *gr = new TGraphErrors(nPts, xPts, yPts, NULL, yErr);
  gr->SetMarkerColor(kRed); gr->SetMarkerStyle(24); gr->SetMarkerSize(0.8);

  TF1 *f = new TF1("f", myFunction, 0.0, 1.0, 2);
  f->SetParNames("q","m"); f->SetParameters(0.0, 0.1);
  gr->Fit(f,"R0");

  TCanvas *c = new TCanvas("c", "This is a canvas", 500, 500);
  c->SetLeftMargin(0.15); c->SetBottomMargin(0.15); c->cd();
  TH2F *h = new TH2F("h", "Title", 10,-0.02,1.15,10,-0.05,1.15);
  h->GetXaxis()->SetTitle("x-title"); h->GetYaxis()->SetTitle("y-title");
  h->SetStats(kFALSE); h->Draw("");
  gStyle->SetOptFit(1111);
  gr->Draw("P,same");
  f->Draw("l,R,same");
  return 0;
  }
```

Fit operation



Fit result drawn on a Canvas

info in the command line as for TH1F

Important tools not presented

- There are many other tools offered by ROOT
  not presented here
  very important for real data analyses

- ROOT files (TFile Class Reference)
  used for the storage and reading of ROOT objects like TH1F, TH2F, TGraph, TGraphErrors, TF1, TTree etc..

- ROOT Trees (TTree)
  used to store and read data
  very efficient for drawing and comparing histograms using the ROOT interpreter..

- ROOT TBrowser (TBrowser Class Reference)
  to quickly visualize object saved in a ROOT file from the ROOT interpreter

see ROOT documentation and many resources that can be found online..

Conclusions

The replica method is an interesting and powerful tool to explore statistical correlations and construct uncertainty bands

playing with the replica requires the knowledge of different tools for data analysis

equivalently, it can be used to better understand the tools

A framework for the implementation of the replica method is offered by ROOT

not simple.. yes, but good practice allows to become familiar with it!

let’s start with an exercise!

For any question, feel free to contact me:
albi.kerbizi@ts.infn.it

Exercise: CI for the "tensor charge"

- Distribution of the «tensor charge» and evaluation of the CI corresponding at 68% and 90% CL

«tensor charge» fundamental property of the nucleon
can be evaluated in lattice QCD → important contact point between phenomenological extractions of transversity and the lattice

Exercise: CI for the "tensor charge"

- Distribution of the «tensor charge» and evaluation of the CI corresponding at 68% and 90% CL

  the (contribution of u-quarks to the) tensor charge g_T^u is given by

$$g_T^u = \int_{x_{min}}^{x_{max}} h(x)\ dx$$

  the range of integration should be between 0 and 1
  here data range limited → truncated contribution to the tensor charge, in the range

$$x_{min} = 0.008, x_{max} = 0.21$$

Exercise: CI for the "tensor charge"

- Distribution of the «tensor charge» and evaluation of the CI corresponding at 68% and 90% CL

  the (contribution of u-quarks to the) tensor charge g_T^u is given by

  $$g_T^u = \int_{x_{min}}^{x_{max}} h(x)\ dx$$

  the range of integration should be between 0 and 1
  here data range limited → truncated contribution to the tensor charge, in the range

  $$x_{min} = 0.008, x_{max} = 0.21$$

- Using the file «ReplicaMethod_HandsOn.C» (see indico), divide the exercise into steps
  1.a. Construct a TF1 representing h(x) for each replica
  1.b. Construct a TH1F and will with the values of g_T^u
  note: to evaluate the integral of f, with f being a TF1, use
  ```
  double gTu = f->Integral(xmin, xmax);
  ```
  2. Find the CI corresponding to 68% CL and 90%CL
  either use the class ConfidenceInterval defined at the beginning of the ReplicaMethod_HandsOn.C file
  or construct a loop over the histogram's bins your way..

Exercise: CI for the "tensor charge"

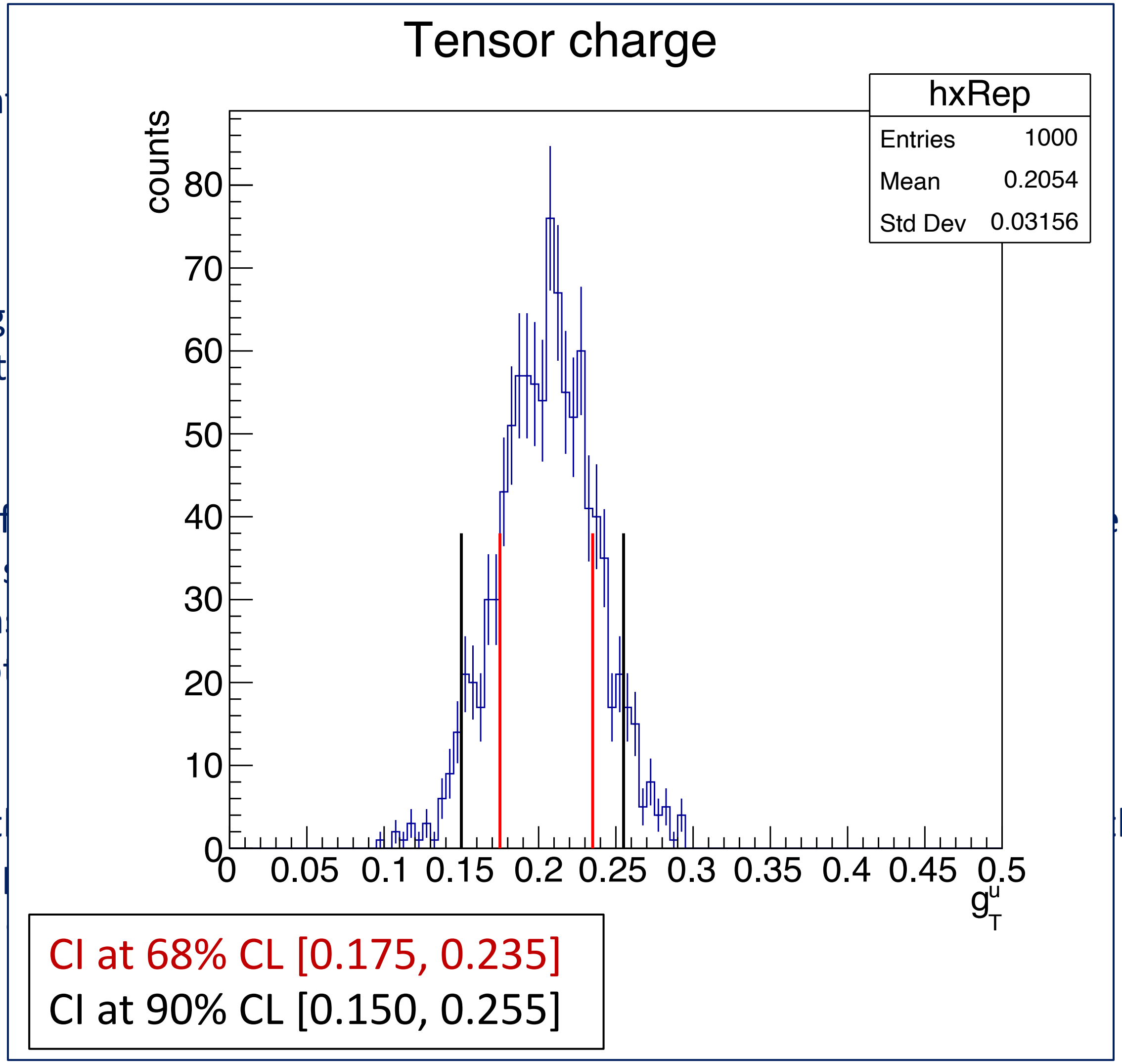
- Distribution of the «tensor charge» and evaluation of the CI corresponding at 68% and 90% CL

  the (con

  the rang
  here dat … the range

- Using the f … into steps
  1.a. Con
  1.b. Con
  not

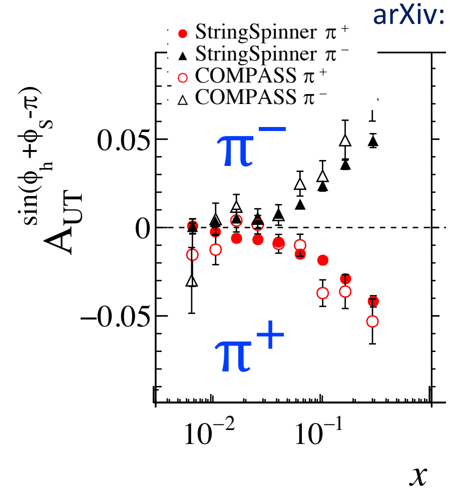
  2. Find
  eit … the
  Re
  or



Backup

One final slide: about my research

Kerbizi, Lönnblad
arXiv: 2305.05058

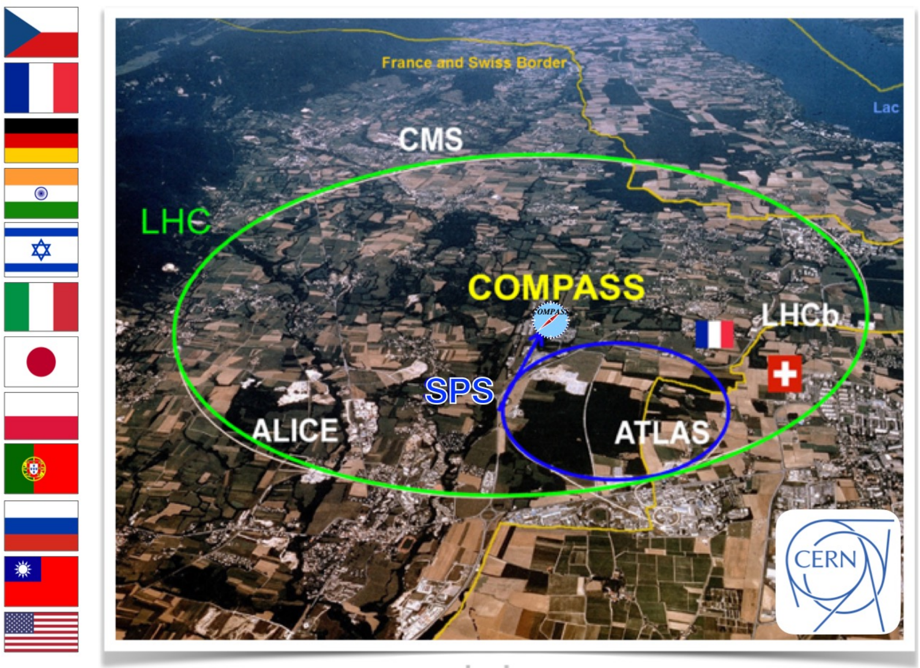


My main research activities

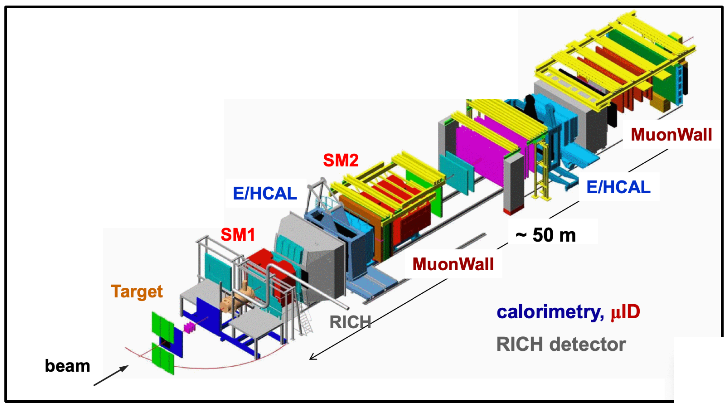
i) modeling and simulation of hadronization using the string fragmentation model, with the goal of reproducing SIDIS and e+e- data

the model: Kerbizi, Artru, Belghobsi, Bradamante, Martin, PRD 97, 074010 (2018)
Kerbizi, Artru, Martin, PRD 104, 114038 (2021)

ii) analysis of SIDIS data at COMPASS



- fixed target experiment at the CERN SPS
  24 institutions, 13 countries
- taking data 2002 – 2022!
  SIDIS and much more..



Many data analyses possible for students!