ROOT Project - part 1

Narges Khorshidi 400170041

Department of Physics at Sharif University of Technology

Winter - Spring 2025

General Description:
In this project, was used Anaconda to configure the Python kernel in Jupyter Notebook and import the ROOT to execute the codes. We have 5 question that i explain about them.

---

1. Sample Generation

Generate a sample of 50 measurements according to a Gaussian p.d.f. with parameters $\mu = 0.2$ and $\sigma = 0.1$.

In this part, a sample of 50 random numbers from a Gaussian (normal) distribution with mean μ=0.2 and standard deviation σ=0.1 is generated and its histogram is plotted.

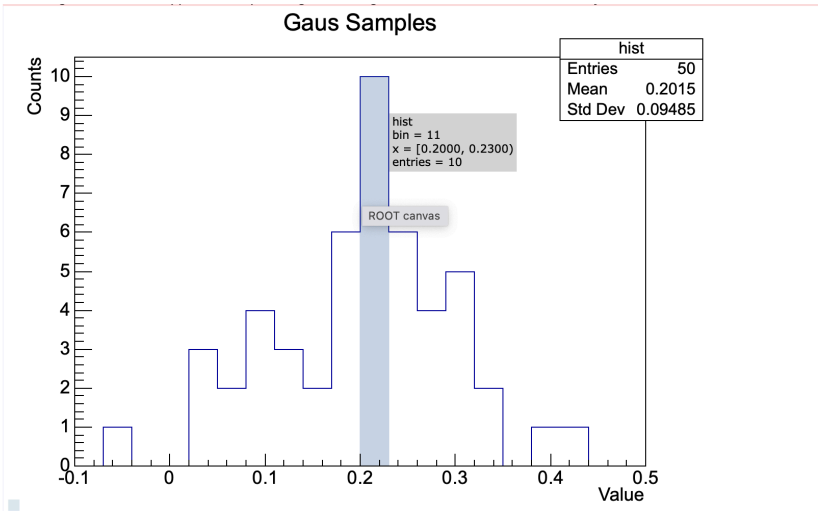
[11]:
```
import ROOT

mu = 0.2
sigma = 0.1
n_samples = 50


samples = [ROOT.gRandom.Gaus(mu, sigma)  for _ in range(n_samples)]

hist = ROOT.TH1F("hist", "Gaus Samples;Value;Counts", int(20), float(-0.1), float(0.5))


for val in samples:
    hist.Fill(val)

c1 = ROOT.TCanvas()
hist.Draw()
c1.Draw()
```

At first line we import ROOT module to generate random data and draw a histogram.

Then We define Gaussian parameters with the mean and variance given in the question for 50 samples in line 2,3 and 4.

The next line Generate 50 random numbers with Gaussian (normal) distribution using the Gaus() function from the ROOT library. This data is stored in the samples list. in line 5

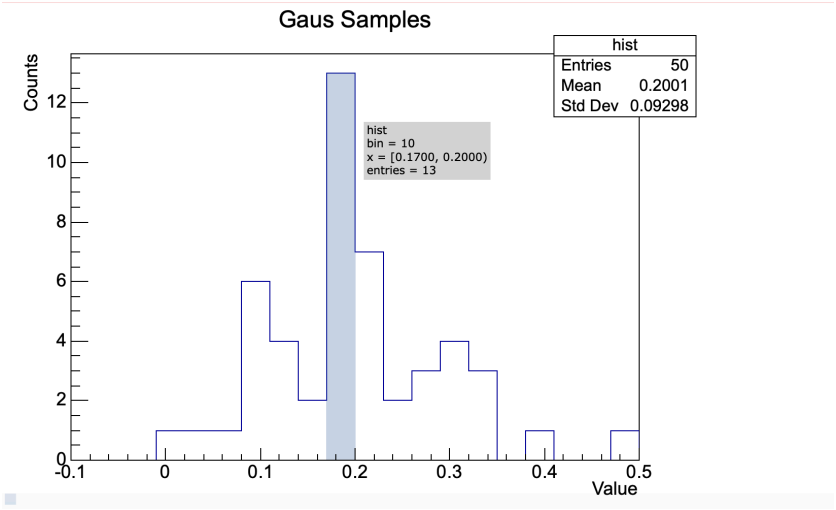
Then in lines 6-11, In the next lines, we define our chart with 20bins and a specific x-axis range, fill the histogram with sample values, and provide a space for drawing it (canvas).



The frequency around the mean is the highest, which shows that the simulation worked correctly.
We generated fifty samples with a mean of 0.2 and a variance of 0.1.

The horizontal axis(x) shows the values of the random samples in the interval of the axis. And the vertical axis(y) shows the frequency, or number of samples in each interval.

And the gray box for each interval shows the number and interval and the number of intervals.



We have another output.
in both outputs, we see the mean is close to 0.2 and the variance is close to 0.1.

2. ML estimation

With the obtained data sample, find the ML estimations of the mean and the variance of the parent distribution.

We are give a sample of data drawn from a Gaussian distribution.
we compute:
The Maximum Likelihood Estimate (MLE) of the mean (μ)
The MLE of the variance ($\sigma 2$)

[36]:
```
import ROOT

mu = 0.2
sigma = 0.1
n_samples = 50

samples = [ROOT.gRandom.Gaus(mu, sigma) for _ in range(n_samples)]

hist = ROOT.TH1F("hist", " Guassian Sample;Val;Counts", 20, -0.1, 0.5)

for val in samples:
    hist.Fill(val)

c1 = ROOT.TCanvas("c1", "Canvas", 800, 600, )
hist.Draw()

mu_mle = hist.GetMean()
sigma2_mle = hist.GetRMS()**2

print(f"MLE Estimate of Mean (mu): {mu_mle:.4f}")
print(f"MLE Estimate of Variance (sigma^2): {sigma2_mle:.6f}")

c1.Update()
```

```
MLE Estimate of Mean (mu): 0.1934
MLE Estimate of Variance (sigma^2): 0.008124
```

Like previous part at first we generates 50 samples from a Gaussian distribution with:
True mean μ=0.2

True standard deviation σ=0.1

and we Fills it with the 50 generated data points.

Then the hist.GetMean() gives the sample mean → MLE of μ

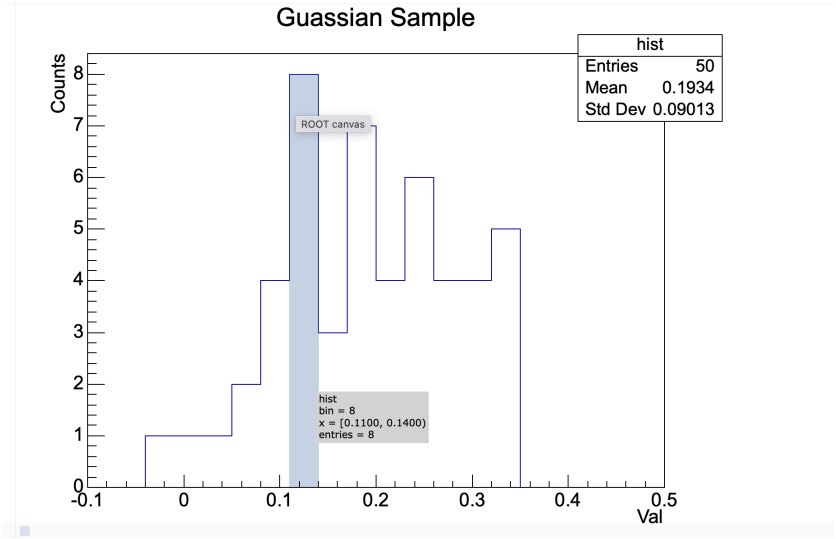
And the hist.GetRMS() gives the sample standard deviation (i.e., root mean square) → MLE of σ

Then it squares it to obtain the variance $\sigma 2$
These are the Maximum Likelihood Estimators:

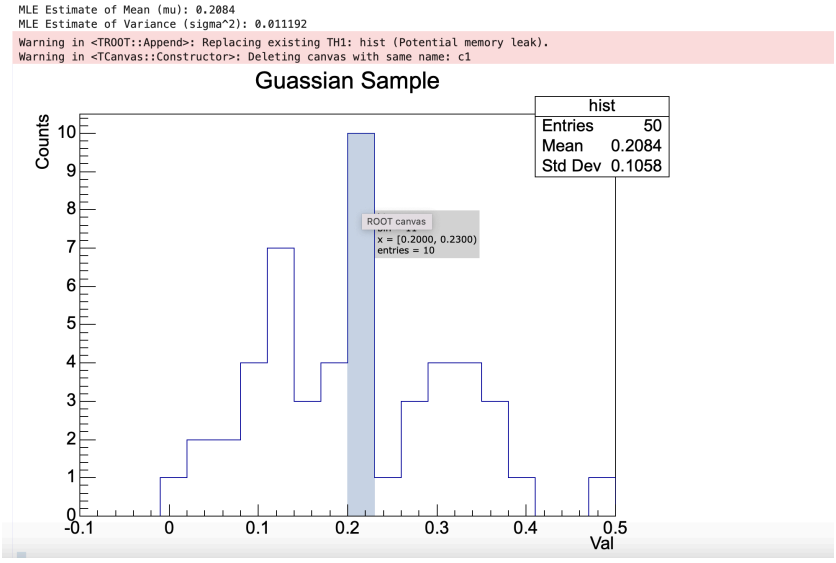
$$\hat{\mu}_{\text{MLE}} = \frac{1}{n} \sum x_i$$

$$\hat{\sigma}^2_{\text{MLE}} = \frac{1}{n} \sum (x_i - \mu)^2$$

At the end we Prints the computed MLE estimates from the data sample.



In the diagram above the horizontal axis (x-axis) shows the sample values (ranging approximately from -0.1 to 0.5).
The vertical axis (y-axis) shows the frequency (i.e., the number of times each value occurs).
The data are centered around the mean value $\mu \approx 0.1934$, which is consistent with a normal distribution.
The gray rectangle in the center-bottom shows the information for a specific bin:
Bin number: 8
Range: $[0.1100, 0.1400]$
Number of entries in this bin: 8 observations



This is the same output for this code.

3. true parameter values

Plot the Gaussian distribution using the parameter values for which the likelihood function (and hence also its logarithm) are a maximum. In the same plot, also show the p.d.f. using the true parameter values..

```
import ROOT

mu = 0.2
sigma = 0.1
n_samples = 50

samples = [ROOT.gRandom.Gaus(mu, sigma) for  _ in range(n_samples)]

hist = ROOT.TH1F("hist", "Gaussian Samples;Value;Count", 20, -0.1, 0.5)

for val in samples:
    hist.Fill(val)

mu_mle = hist.GetMean()
sigma = hist.GetRMS()
sigma2_mle = sigma**2

print(f"MLE Estimate of Mean (mu): {mu_mle:.4f}")
print(f"MLE Estimate of Variance (sigma^2): {sigma2_mle:.6f}")

c1 = ROOT.TCanvas("c1", "Canvas", 800, 600)
hist.Draw("E")

gaus_mle = ROOT.TF1("gaus_mle", "gaus", -0.1, 0.5)
gaus_mle.SetParameters(hist.GetMaximum(), mu_mle, sigma)
gaus_mle.SetLineColor(ROOT.kRed)
gaus_mle.SetLineWidth(2)



gaus_true = ROOT.TF1("gaus_true", "gaus", -0.1, 0.5)
gaus_true.SetParameters(hist.GetMaximum(), mu, sigma)
gaus_true.SetLineColor(ROOT.kBlue)
gaus_true.SetLineWidth(2)
gaus_true.SetLineStyle(2)


gaus_mle.Draw("SAME")
gaus_true.Draw("SAME")

c1.Update()
```

In the first 4 lines: This creates 50 random numbers from a normal distribution with μ=0.2 and σ=0.1. These are our simulated experimental data points.

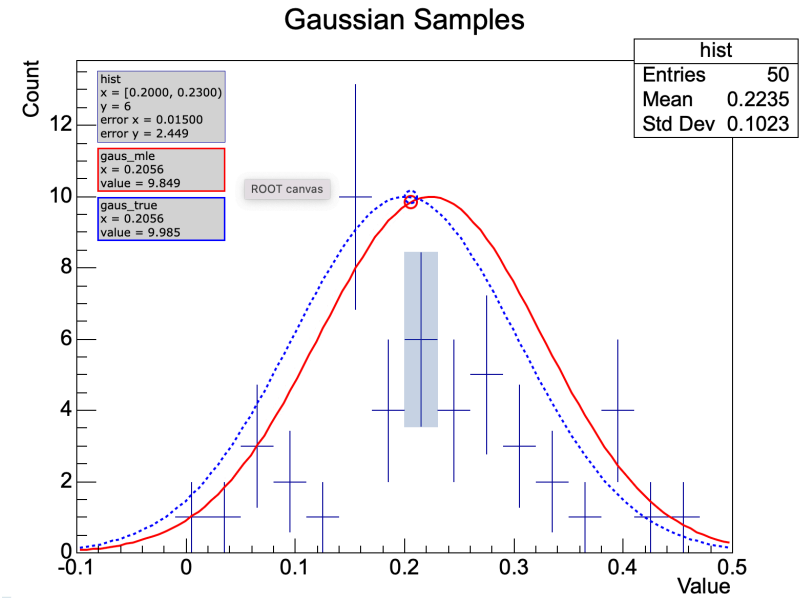
Then in the next lines: Fills a histogram with the generated data.
Uses 20 bins spanning from -0.1 to 0.5.

Next: Mean (GetMean): is the MLE for μ
RMS (GetRMS): is the MLE for σ
These are the values that maximize the likelihood function for a Gaussian.

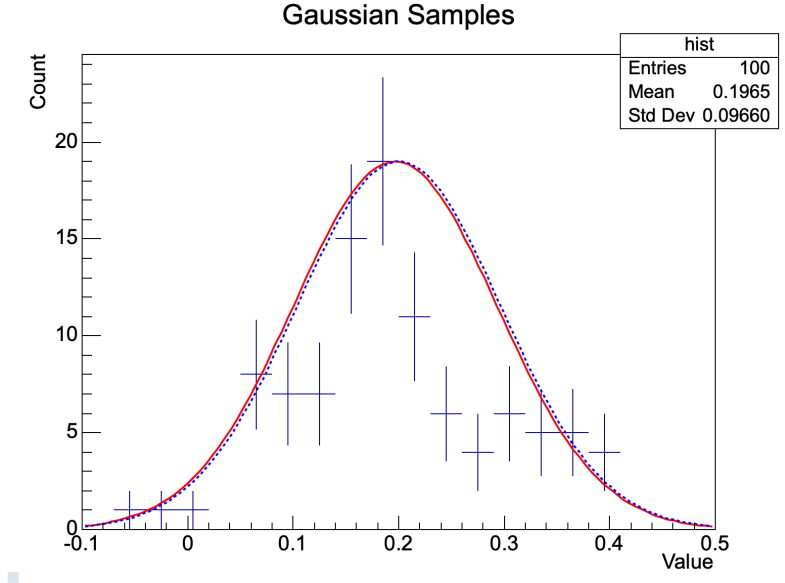
Next: Defines a Gaussian using the estimated parameters from data (MLE). hist.GetMaximum() is used for normalization (y-scale).

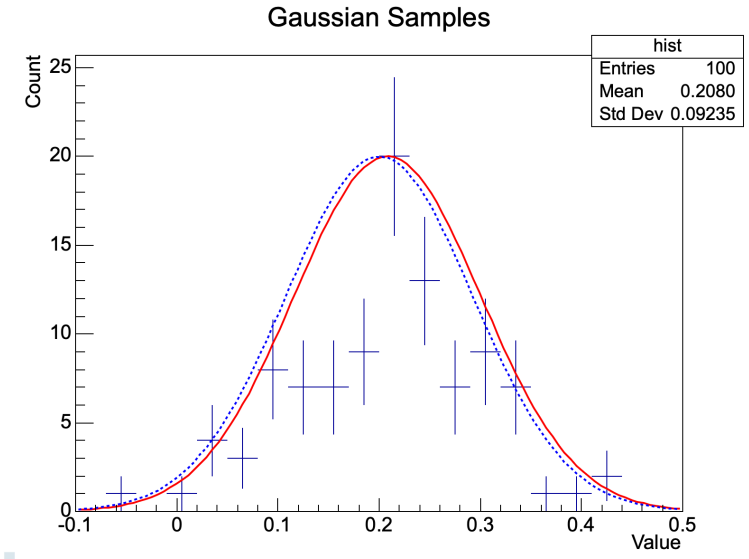
Next: The original Gaussian PDF using the true parameters used in generating the data.

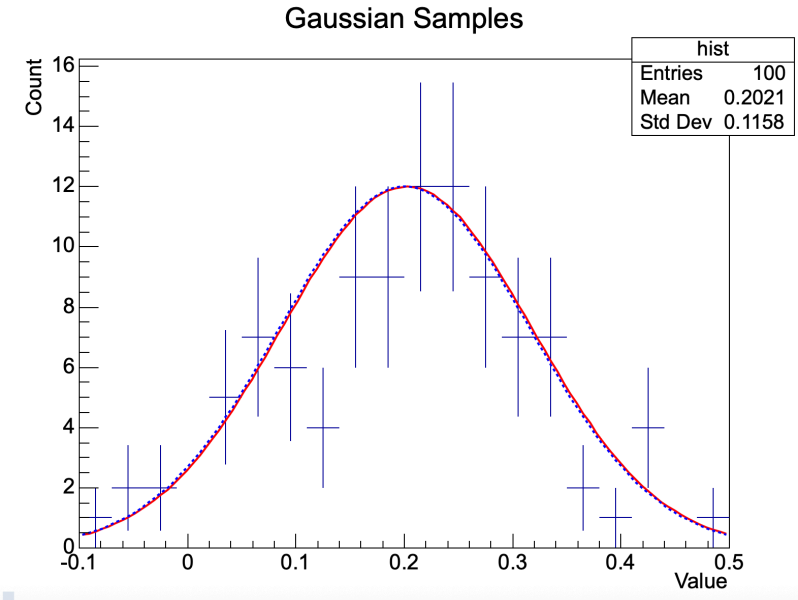
At the end: Draws the histogram with error bars (”E”) and both Gaussian curves overlaid on top.



Red curve = Gaussian using MLE estimates (μ MLE = 0.2056) Blue dotted curve = Gaussian using the true values (μ true=0.2056)
Histogram = sample distribution
Entries = 50 means 50 samples used
Mean = 0.2235, Std Dev = 0.1023 are summary statistics from the histogram
The red and blue curves are very close — this shows that the MLE is a good estimator of the true parameters.
Another point is that as the data increases, the peaks of the two charts coincide with each other.







Data: Generated with real parameters
MLE: Obtained from the data using the maximum likelihood method
Charts: Two Gaussian probability density functions plotted: one with real parameters, one with MLE

The goal of the question is to compare the shape and estimated parameters with the real values —> visible in the chart.

4. Monte Carlo method

Estimate the variance of the mean with the Monte Carlo method. To do this one must simulate a large number of experiments, compute the ML estimates each time and look at how the resulting values are distributed.
Regarding the first Monte Carlo experiment as the 'real' one, simulate 1000 further experiments with 50 measurements each (i.e. repeat the first bullet). For the 'true' parameters in the Monte Carlo program, the estimated values from the real experiment can be used (i.e. those obtained from the second bullet above).
Plot the histogram of the resulting ML estimates. The sample standard deviation from the 1000 experiments must be calculated from the unbiased estimator of the variance of the p.d.f. (i.e. s2).

I looked for a more realistic example to get more insight into this question. For example, in simple terms, in expensive experiments like CMS or ATLAS at CERN, we only have one measurement (like a real experiment), but to estimate the statistical error, simulated data is generated thousands of times (with the estimated parameters the first time). Then, it is checked how much dispersion the parameter estimate (like the mass of a particle) has. That is, exactly like what we will do in this question:

Get an estimate of the parameter from the real data, create data with that parameter, and see how much the estimates fluctuate.

In this question, we used ROOT for plotting and numpy for numerical calculations.
First, we set the correct parameters. We obtained 50 samples from a Gaussian distribution with real values.
We simulate the real experiment with 50 samples from a Gaussian distribution with the real parameters that we generated. Then, the values are filled in the histogram. The mean and RMS are taken as MLE estimates from the real data.
In Monte Carlo simulation, the experiment is repeated a thousand times. In each simulation, 50 samples are taken using the estimated parameters mu-mle-real and sigma-mle-real and their average is stored as MLE.
Then the distributions are plotted. And finally we obtain the unbiased variance.

```python
import ROOT
import numpy as np

mu_real = 0.2
sigma_real = 0.1
n_samples = 50

samples_real = [ROOT.gRandom.Gaus(mu_real, sigma_real) for _ in range(n_samples)]

hist_real = ROOT.TH1F("hist_real", "Real Experiment", 20, -0.1, 0.5)
for val in samples_real:
    hist_real.Fill(val)

mu_mle_real = hist_real.GetMean()
sigma_mle_real = hist_real.GetRMS()

print(f"Real experiment:")
print(f" MLE mu = {mu_mle_real:.4f}")
print(f" MLE sigma = {sigma_mle_real:.4f}")

n_MC = 1000
mu_mle_list = []

for i in range(n_MC):

   samples_MC = [ROOT.gRandom.Gaus(mu_mle_real, sigma_mle_real) for _ in range(n_samples)]
    mu_mle_MC = np.mean(samples_MC)
    mu_mle_list.append(mu_mle_MC)

hist_MC =ROOT.TH1F("hist_MC", "Distribution of MLE mu estimates;mu_MLE;Counts", 50, mu_mle_real - 0.05, mu_mle_real + 0.05)

for mu_val in mu_mle_list:
    hist_MC.Fill(mu_val)

c2 = ROOT.TCanvas("c2", " Monte Carlo", 800, 600)
hist_MC.Draw()
c2.Update()

mu_mle_array = np.array(mu_mle_list)
mean_mu_MC = np.mean(mu_mle_array)

s2 = np.sum((mu_mle_array - mean_mu_MC)**2) / (n_MC - 1)
s = np.sqrt(s2)

print(f"\nMonte Carlo Results:")
print(f" Mean of MLE mu estimates = {mean_mu_MC:.6f}")
print(f" Unbiased Variance (s^2) = {s2:.8f}")
print(f" Unbiased Std Dev (s) = {s:.8f}")
```

The x-axis represents the different mean values obtained from each Monte Carlo experiment.

The y-axis represents the frequency of the mean observed in each interval.

The distribution is bell-shaped and almost normal — which is to be expected since we are taking the mean of samples from a Gaussian distribution.

The peak of the graph is around 0.20 (corresponding to the Mean)
The width of the distribution (standard deviation) is about the same as what is reported as Std Dev.
And the gray box at the end shows the number of means in that range.

Mean: The mean of a thousand μ values

Entries: That is, 1000 ˆ values are recorded in this histogram (i.e. 1000 simulations were performed)

entrise: The frequency in each bin

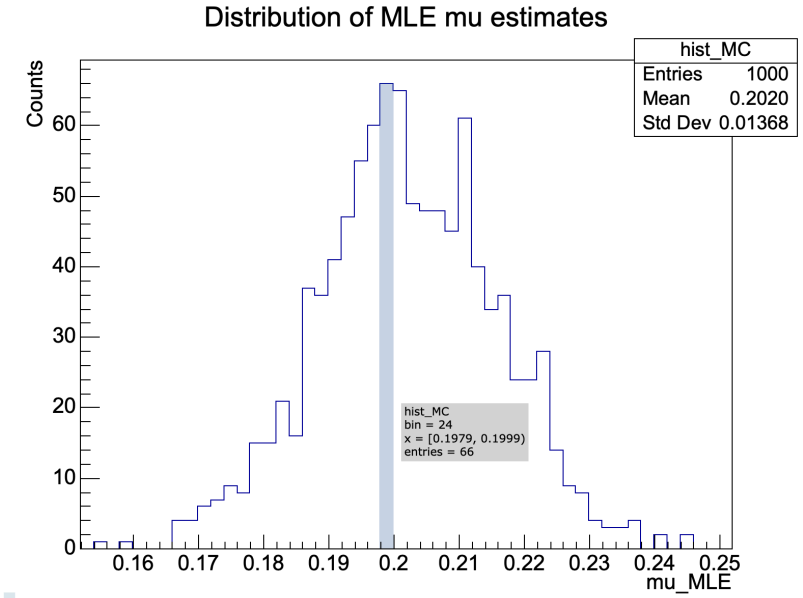
std Dev: Standard deviation (s) — which is the statistical error in estimating the mean μ

```
Real experiment:
 MLE mu = 0.2019
 MLE sigma = 0.0978

Monte Carlo Results:
 Mean of MLE mu estimates = 0.202037
 Unbiased Variance (s^2) = 0.00018723
 Unbiased Std Dev (s) = 0.01368331
```

MLE mu: Mean of 50 data points from the "real experiment" (using MLE)

Mean of MLE mu estimates: Mean of 1000 MLE estimates of μ in Monte Carlo experiments

and for another parameter like these.

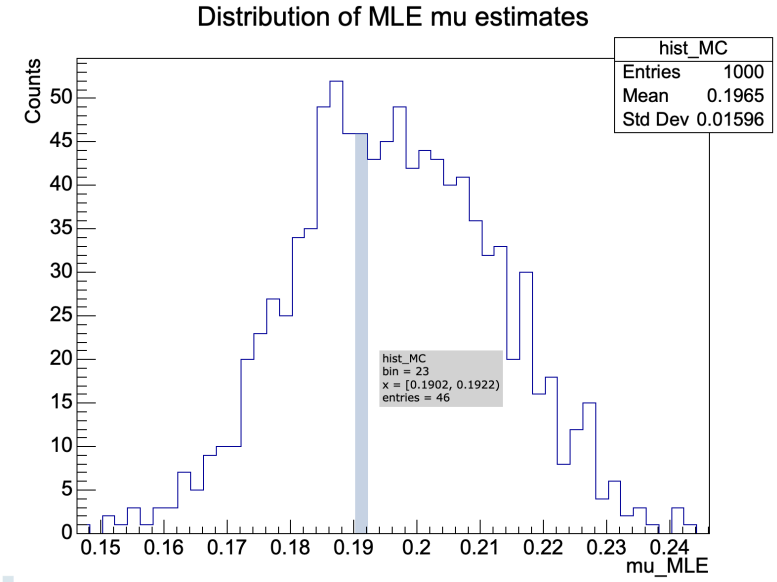


In this figure, as explained in the previous section, the number of each bin is displayed in the figure, and the interval and frequency of the means within that bin.

```
Monte Carlo Results:
 Mean of MLE mu estimates = 0.202037
 Unbiased Variable (s^2) = 0.00018723
 Unbiased Std Dev (s) = 0.01368331) --> Statistical error from MC

Analystical Result:
 Analytical sigma_mu = 0.01382794

Ratio (MC / Analytical) = 0.9895
```

This figure is related to the output of the code for the next question for this specific output in this section. I will explain it.

```
Real experiment:
 MLE mu = 0.1962
 MLE sigma = 0.1118

Monte Carlo Results:
 Mean of MLE mu estimates = 0.196544
 Unbiased Variance (s^2) = 0.00025745
 Unbiased Std Dev (s) = 0.01604520
```




```
Monte Carlo Results:
 Mean of MLE mu estimates = 0.196544
 Unbiased Variable (s^2) = 0.00025745
 Unbiased Std Dev (s) = 0.01604520) --> Statistical error from MC

Analystical Result:
 Analytical sigma_mu = 0.01580861

Ratio (MC / Analytical) = 1.0150
```

According to the explanation of the previous output, we also see everything for another output in this image.

5. Report error

Report the statistical error on the mean found from the Monte Carlo method (fourth bullet above), and compare it with the analytical solution (i.e. $\hat{\sigma} = \frac{\hat{\mu}}{\sqrt{n}}$).

```
mu_mle_array = np.array(mu_mle_list)
mean_mu_MC = np.mean(mu_mle_array)

s2 = np.sum((mu_mle_array - mean_mu_MC)**2) / (n_MC - 1)
s = np.sqrt(s2)

sigma_mu_analytical = sigma_mle_real / np.sqrt(n_samples)

print(f"\nMonte Carlo Results:")
print(f" Mean of MLE mu estimates = {mean_mu_MC:.6f}")
print(f" Unbiased Variable (s^2) = {s2:.8f}")
print(f" Unbiased Std Dev (s) = {s:.8f}) --> Statistical error from MC")

print(f"\nAnalystical Result:")
print(f" Analytical sigma_mu = {sigma_mu_analytical:.8f}")

ratio = s / sigma_mu_analytical
print(f"\nRatio (MC / Analytical) = {ratio:.4f}")
```

mu-mle-list is a list of 1000 ˆ values that we got from Monte Carlo. With np.mean() we take the average of all the estimates → this is our "final estimate of ".
S2 is the unbiased variance of the estimates of μ
s is the standard error of the mean that we get from the simulation
This error comes from the Monte Carlo data itself (exactly what the question asked for).
sigma-mle-real is the ˆ that comes from the real experimental data (i.e. the real RMS) n-samples is the number of observations (e.g. 50) With this formula, we get the analytical value of on the mean μ
Lines 6 to 9 show what the average estimates are and what the Monte Carlo error is.
The next two lines show the theoretical error on the mean μ.
And finally, comparing these two values.

```
Monte Carlo Results:
 Mean of MLE mu estimates = 0.202037
 Unbiased Variable (s^2) = 0.00018723
 Unbiased Std Dev (s) = 0.01368331) --> Statistical error from MC

Analystical Result:
 Analytical sigma_mu = 0.01382794

Ratio (MC / Analytical) = 0.9895
```

Mean of MLE mu estimates = The average of 1000 MLE estimates of — this approximates the expected value ˆ from Monte Carlo.
s2 = The unbiased sample variance of the 1000 estimates.
s = The square root of variance — this is the statistical error (standard deviation) on ˆ estimated from Monte Carlo.

Analytical Result: Analytical sigma-mu = 0.01382794
This was computed using the analytical formula:

$$\sigma_{\bar{\mu}} = \frac{\hat{\sigma}}{\sqrt{n}} = \frac{0.0978}{\sqrt{50}} \approx 0.01383$$

- $\hat{\sigma}$ is the MLE sigma or RMS from the real experiment (i.e., `sigma_mle_real`).
- $n = 50$ is the number of samples per experiment.

Ratio (MC / Analytical) = 0.9895
Ratio:

$$\frac{\text{MC Std Error}}{\text{Analytical Std Error}} = \frac{0.01368}{0.01383} \approx 0.9895$$

This means the Monte Carlo estimated error is only about **1% smaller** than the analytical value.

The difference is **very small and acceptable**, confirming the **accuracy of the simulation**.

```
Monte Carlo Results:
 Mean of MLE mu estimates = 0.196544
 Unbiased Variable (s^2) = 0.00025745
 Unbiased Std Dev (s) = 0.01604520) --> Statistical error from MC

Analystical Result:
 Analytical sigma_mu = 0.01580861

Ratio (MC / Analytical) = 1.0150
```

Aand this is the output of the second example in the previous question.