

(Exercises - day 1)

Introduction to afternoon exercises

- 'Seeing is believing' → Aim to make you work hands-on with some of the concepts covered this afternoon
- 'Lightweight' → All exercises are *easy to execute*
 - Much of the heavy lifting is done with prepared code that uses existing tools to the hard stuff (ROOT, RooFit)
 - The goal is that you spend as little time as possible learning software & custom tools, *but a tiny bit of learning is inevitable*: most statistics concepts of interest are too complex to code up from scratch...
 - You start from already functional input files → very limited coding required! Choice of language: C++ or Python. *All exercises exist in 2 versions.*
- Software & computing setup
 - *Prepared setup on 'stoomboot' (stbc-i1/i2/i3) and 'callysto'*. Nothing is CPU or network intensive → connecting from your laptop should work fine
 - Environment setup for a standard ROOT setup provided, but you can use your own if you like (most if not all exercises are not version-dependent)
 - Working on your own laptop is also fine (if you have ROOT installed already and know how to use it) If not → use provided setup

Setup 1 – Prepared installation on ‘stoomboot’ - Easy

- Step 0 – Login into stoomboot and **create a dedicated directory** for today's exercises
- Step 1 – retrieve the **setup & exercise guide** for today
 - web: http://www.nikhef.nl/~verkerke/stats2024/day1/cpp/stats2024_exercises_day1.pdf
 - file: `/user/verkerke/stats2024/day1/cpp/stats2024_exercises_day1.pdf`
- Step 2 – follow the instructions on page 1 of the guide to **setup the software** on stoomboot
 - `source /cvmfs/sft.cern.ch/lcg/app/releases/R00T/6.32.08/x86_64-almalinux9.4-gcc114-opt/bin/thisroot.sh`
- Step 3 – **Copy the input files** for today's exercises
 - web: <http://www.nikhef.nl/~verkerke/stats2024/day1/cpp/> or <http://www.nikhef.nl/~verkerke/stats2024/day1/python/>
 - file: `/user/verkerke/stats2024/day1/cpp/` or `/user/verkerke/stats2024/day1/python/`

Setup 1 – Prepared installation on ‘stoomboot’ - Easy

- Instructions are in the pdf file in the exercises directory
- First part in gray boxes tells you
 1. Where are the input files and the software
 2. A 5 minute primer to RooFit model building. Most of of this is even only for background as all exercises start with something working already.

Topical Lectures Statistics – Exercises Set 1

Wouter Verkerke (Dec 2022)

General Instructions

Input files

Input files for exercises can be found in three places

1. At nikhef.stbc-1.nikhef.nl in directory `~verkerke/stats2022/`
2. At CERN (lplus7.cern.ch) in directory `~verkerke/public/stats2022`
3. On the web at <http://www.nikhef.nl/~verkerke/stats2022>

Running ROOT

All exercises are based on ROOT. You are recommended to use version 6.24.08.

In which all prepared material has been tested. To pick up a pre-installed ROOT

version please execute the following setup script

```
source ~/cvfms/sft.cern.ch/cgi/app/releases/ROOT/6.24.08/x86_64-centos7-gcc48-opt/bin/thisroot.sh
```

This release will work both at Nikhef and at CERN

Where to work

If you have an account at Nikhef, please work on `stbc-1/2/3.nikhef.nl` only and not on `login.nikhef.nl` (these run CentOS – required for above ROOT version)

If you have an account at CERN, you can also work on `lplus7.cern.ch`, this will select a CentOS7 node (required for above ROOT version)

You can also work directly on your laptop if you have ROOT installed yourself

Quick reference guide to RooFit model building

In these tutorial exercises we will use RooFit to build probability models as that allows to rapidly build complex models. This requires some familiarity with the RooFit model building syntax. It is easy enough to pick up ‘on the way’ from input files given with this tutorial, but for completeness this page also presents a quick reference guide to the model building syntax and strategy

RooFit model structure

The key feature of RooFit model building is that all elements of a probability model (functions, pdfs, variables) are all individual objects that connect to each other. For example a Gaussian probability density model is expressed as

```
RooRealVar x("x","x",0,-10,10);
RooRealVar mean("mean","mean",0,-10,10);
RooRealVar sigma("sigma","sigma",1,0.1,10);
RooGaussian gauss("gauss","gauss",x,mean,sigma);
```

The simplest strategy to build these models is to have all objects contain in a workspace, and use the ‘factory’ tool to fill the workspace with objects:

```
RooWorkspace w("w");
w.factory("Gaussian::gauss(x[0,-10,10],mean[0,-10,10],sigma[3,0.1,10])");
w.print(); // see what's inside the workspace
```

Inside the workspace this factory specification builds exactly the model shown above. The elements of the model can be accessed as follows:

```
RooAbsPdf* gauss = w.pdf("gauss"); // extract a pdf
RooRealVar* x = w.var("x"); // extract a variable
```

RooFit factory syntax

The factory syntax is quick to learn since it has very few rules

- To create a variable use `x [x1], [x2], [x3]`.
- To create a pdf or function use `(Type::name)(args...)`
Any RooFit function or pdf class can be created this way.
You are allowed to omit the ‘Roo’ prefix from any class name.
The meaning and order of the args match those specified in the constructor of the class (after the mandatory name and title). Example:

```
C++ constructor: RooGaussian gauss("gauss","Gaussian",x,mean,sigma)
Factory spec: w.factory("Gaussian::gauss(x,mean,sigma)");
```
- Any field where an object name is expected must either contain the name of a previously created object, or it can be created on this spot.

```
w.factory("x[0,-10,10]");
w.factory("Gaussian::gauss(x,mean[0,-10,10],sigma[3,0.1,10])");
```

In 2nd line above, for x, a previously created object is referenced, for mean and sigma, new objects are created in place.

Special Factory rules for operators

For certain functions and pdf a custom syntax exists to simplify their use. The most important ones are shown here

- Addition of two (or more) pdfs

```
SUM:ob::Name(frac1*pdf1,ndf2); // shape-only pdf
SUM:ob::Name(yield1*pdf1,yield2*pdf2); // extended pdf. N(exp) => y1+y2
```
- Multiplication of two (or more) pdfs

```
PROD:ob::Name(pdf1,ndf2,...); // product of independent pdfs
PROD:ob::Name(pdf1|x,ndf2,...); // product involving a conditional pdf
```
- Interpreted function expressions (all formula expressions can be used). All symbols reference in the expression must be passed as arguments.

```
expr::funcName('some expression',<List of objects used>);
```

Example use cases of these operators are given in the input files of various exercises and will further clarify their use. More special operator syntax exists, but is not needed for this course (e.g. for convolutions, joint models, integrals, projections, amplitude sums)

Basic use of RooFit models

Probability models in RooFit can be used for event generation, (likelihood) fitting and plotting. All of these are one-line operations.

- Generation of unbinned toy datasets

```
RootDataSet pdf.generate(observables,eventcount);
RootDataSet pdf.generate(observables); // for extended models only
```
- Unbinned Likelihood fit of model to data

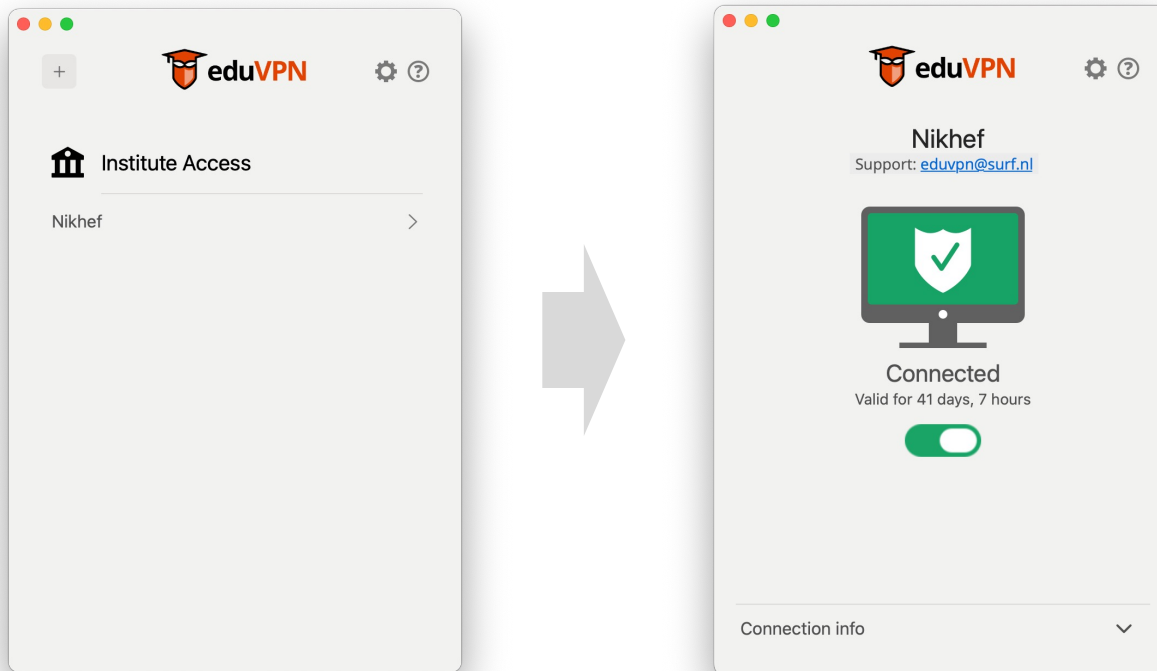
```
pdf.fitTo(data);
RootFitResult r = pdf.fitTo(data,Save()); // save extra info
```
- Plotting of data and model

```
RootPlots frame = obs.frame(); // creates empty plot frame
data.plotIn(frame); // plot data on frame
pdf.plotIn(frame); // project pdf on obs, normalize to data
```

Questions? Don't hesitate to ask!

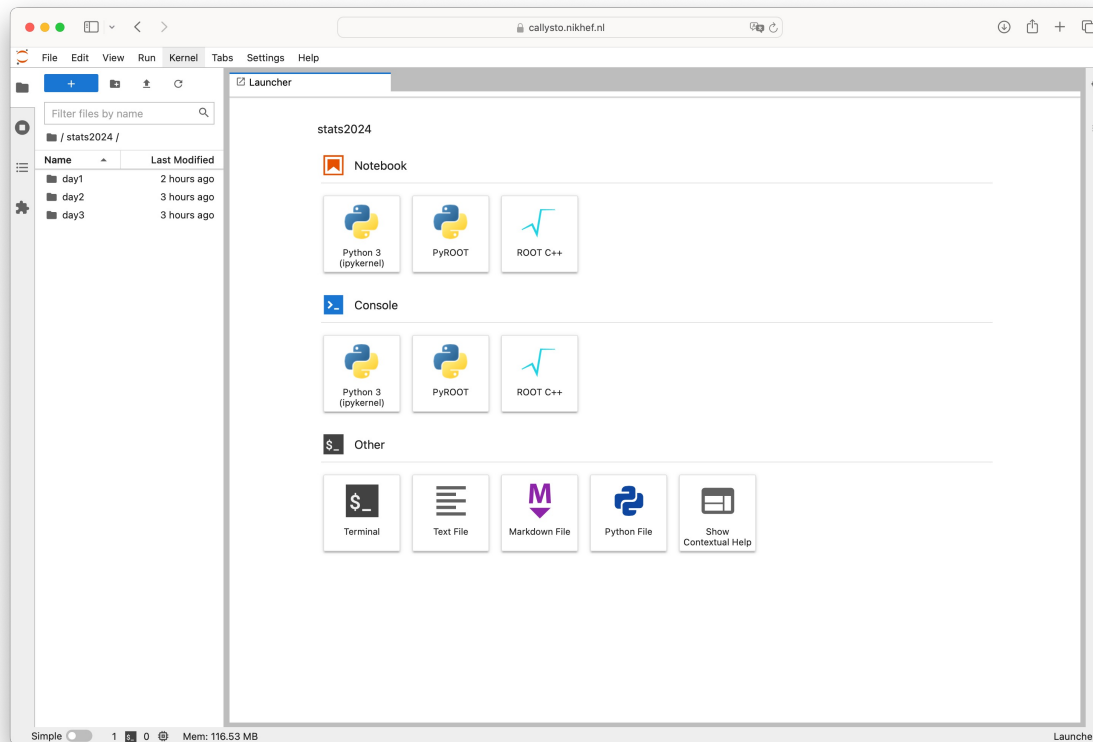
Setup 2 – Jupyter notebook server ‘callysto’ – Easiest!

- Connect with your web browser to the Nikhef Jupyter notebook server: <https://callysto.nikhef.nl>
- Important point #1 – From CWI room Z009 you first need to **activate EduVPN** in ‘Institute Access Mode’



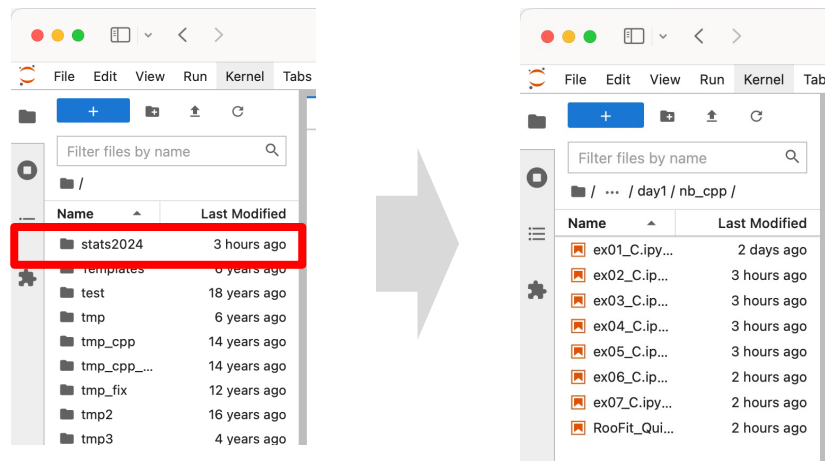
Setup 2 – Jupyter notebook server ‘callysto’ – Easiest!

- Connect with your web browser to the Nikhef Jupyter notebook server: <https://callysto.nikhef.nl>
- Important point #2 – Login with your **Nikhef SSO credentials**, upon success you see this



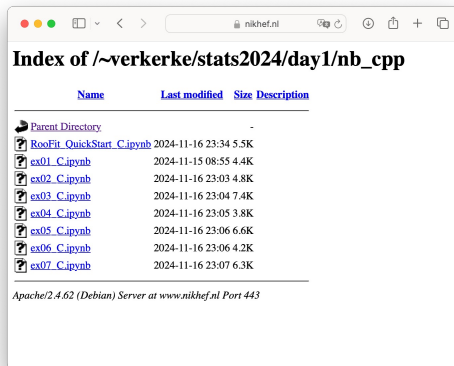
Setup 2 – Jupyter notebook server ‘callysto’ – Easiest!

- Last step: **retrieve the exercises in notebook form**
- **Option 1: login in to login.nikhef.nl** with ssh
From your home directory do the following
 - `mkdir stats2024`
 - `cd stats2024`
 - `cp /user/verkerke/stats2024/day1/nb_cpp/* .`
`cp /user/verkerke/stats2024/day1/nb_python/* .`
- You can see all your files at Nikhef on the callysto system, simply navigate to the directory `stats2024` and you are ready!



Setup 2 – Jupyter notebook server ‘callysto’ – Easiest!

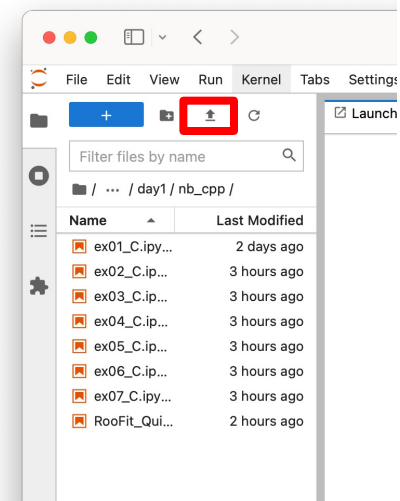
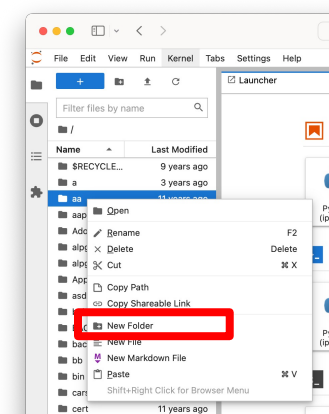
- Last step: **retrieve the exercises in notebook form**
- **Option 2: download / upload via laptop web browser**
Navigate to http://www.nikhef.nl/~verkerke/stats2024/day1/nb_cpp/



Step 1 - Download files to a local directory on your laptop

Step 2a – Create a directory for your files in callysto using the ‘right-click’ menu

Step 2b – Navigate to the new directory by double-clicking on it

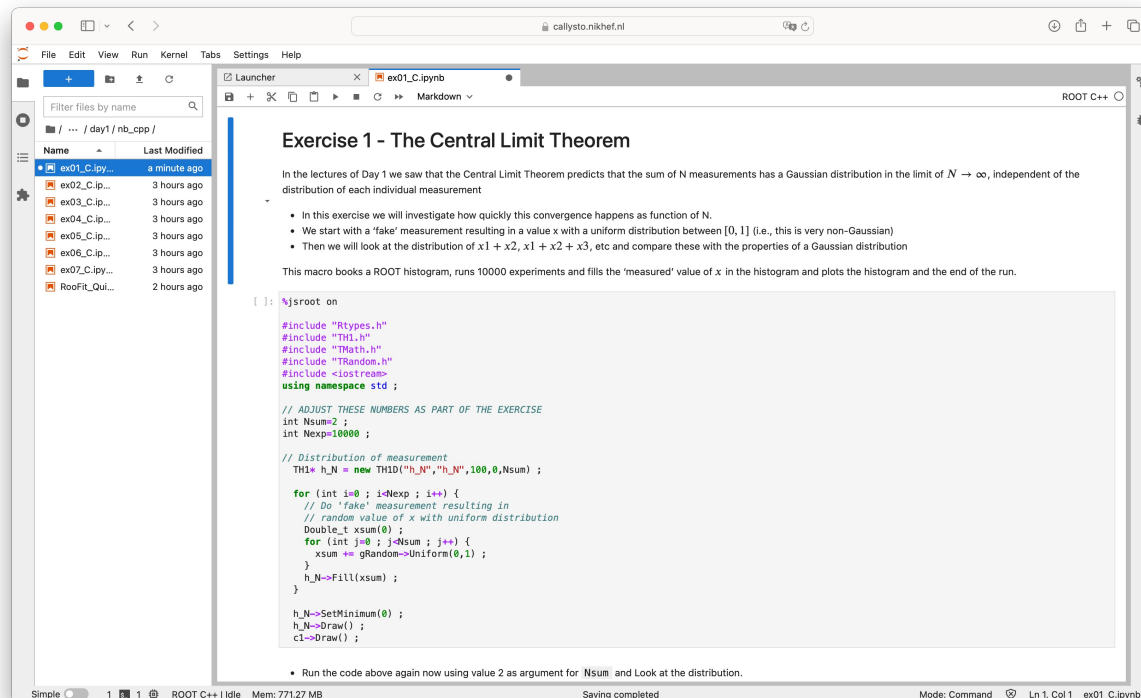


Step 3 – Upload files from your laptop local directory to callysto

Note that all files and directories are actually on login/stoomboot

Running notebooks on callysto

- Double-click on a notebook to open it
- Instructions are *inside* the notebooks
- Notebooks consists of alternating blocks of ‘markdown’ (documentation) and ‘code’ (= executable C++ or python code)



The screenshot shows a Jupyter notebook titled "Exercise 1 - The Central Limit Theorem". The notebook is displayed in a web browser window with the URL "callysto.nikhef.nl". The interface includes a file explorer on the left, a menu bar (File, Edit, View, Run, Kernel, Tabs, Settings, Help), and a main content area. The content area is divided into two sections: a markdown block and a code block. The markdown block contains text about the Central Limit Theorem and a list of bullet points. The code block contains C++ code for a ROOT histogram simulation. The code includes headers for Rtypes, TH1, TMath, TRandom, and TOSTREAM, and defines a macro for generating a histogram of fake measurements. The code is enclosed in a code cell with a prompt character "!" and a cursor.

```
! : \jsroot on

#include "Rtypes.h"
#include "TH1.h"
#include "TMath.h"
#include "TRandom.h"
#include "TOSTREAM"
using namespace std;

// ADJUST THESE NUMBERS AS PART OF THE EXERCISE
int Nsum=2;
int Nexp=10000;

// Distribution of measurement
TH1* h_N = new TH1D("h_N","h_N",100,0,Nsum);

for (int i=0; i<Nexp; i++) {
    // Do "fake" measurement resulting in
    // random value of x with uniform distribution
    Double_t xsum(0);
    for (int j=0; j<Nsum; j++) {
        xsum += gRandom->Uniform(0,1);
    }
    h_N->Fill(xsum);
}

h_N->SetMinimum(0);
h_N->Draw();
c1->Draw();
```

Markdown block

Code block

Running notebooks on callysto

- To **execute code**, click on the code block (blue sidebar will move to it) and click 'Run' button, output appears below

The screenshot shows a Jupyter notebook interface on the Callisto platform. The top toolbar contains a 'Run' button, which is highlighted with a red box and a red arrow pointing to it. The notebook content is as follows:

Exercise 1 - The Central Limit Theorem

In the lectures of Day 1 we saw that the Central Limit Theorem predicts that the sum of N measurements has a Gaussian distribution in the limit of $N \rightarrow \infty$, independent of the distribution of each individual measurement

- In this exercise we will investigate how quickly this convergence happens as function of N .
- We start with a 'fake' measurement resulting in a value x with a uniform distribution between $[0, 1]$ (i.e., this is very non-Gaussian)
- Then we will look at the distribution of $x_1 + x_2$, $x_1 + x_2 + x_3$, etc and compare these with the properties of a Gaussian distribution

This macro books a ROOT histogram, runs 10000 experiments and fills the 'measured' value of x in the histogram and plots the histogram and the end of the run.

```
[2]: %jsroot on
#include "Rtypes.h"
#include "TH1.h"
#include "Math.h"
#include "Random.h"
#include <iostream>
using namespace std;

// ADJUST THESE NUMBERS AS PART OF THE EXERCISE
int Nsum=2;
int Nexp=10000;

// Distribution of measurement
TH1* h_N = new TH1D("h_N","h_N",100,0,Nsum);

for (int i=0; i<Nexp; i++) {
    // Do 'fake' measurement resulting in
    // random value of x with uniform distribution
    Double_t xsum(0);
    for (int j=0; j<Nsum; j++) {
        xsum += gRandom->Uniform(0,1);
    }
    h_N->Fill(xsum);
}

h_N->SetMinimum(0);
h_N->Draw();
c1->Draw();
```

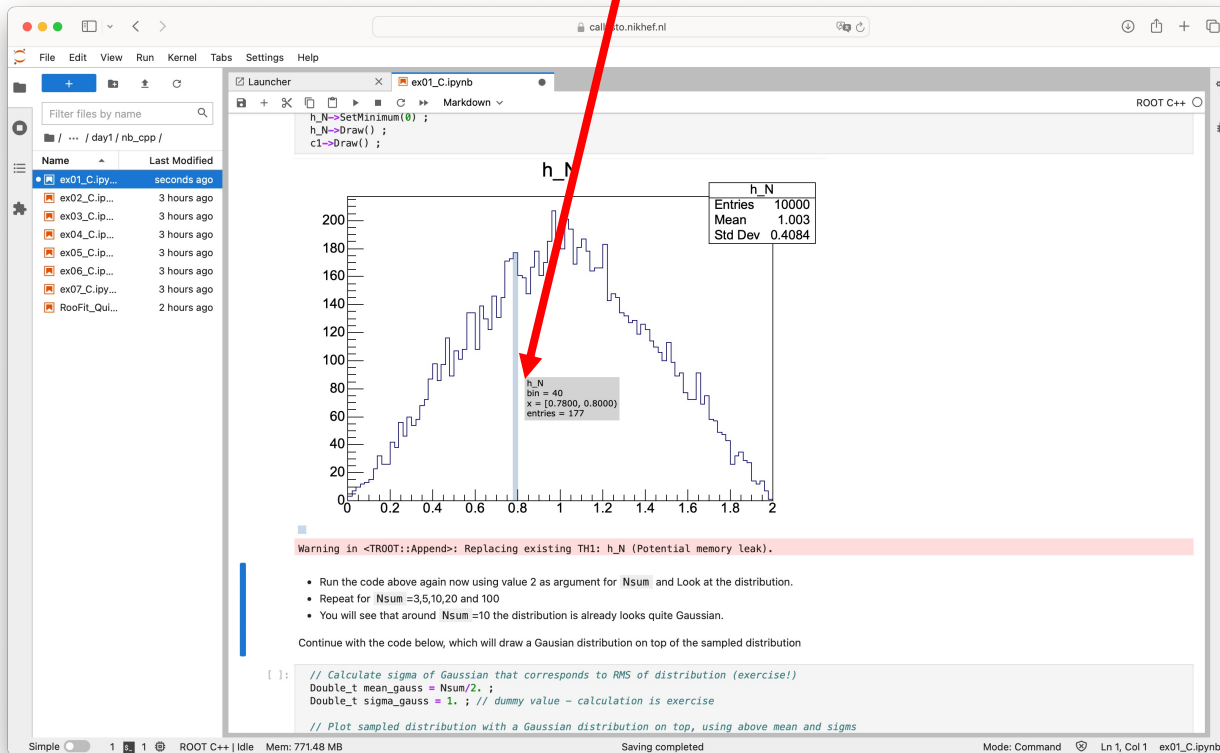
The output is a ROOT histogram plot titled 'h_N'. The x-axis ranges from 0 to 2, and the y-axis ranges from 0 to 200. The histogram shows a Gaussian distribution centered around 1.0. A statistics box next to the plot displays the following values:

h_N	
Entries	10000
Mean	0.9962
Std Dev	0.4058

Output

Running notebooks on callisto

- **Graphical output is interactive** (move mouse over it)



- You can zoom in (and out again) on graphs

Running notebooks on callysto

- You can **modify the code** (and then rerun it), double-click on code block to edit it.
- To (re)run it – either press ‘run’ button, or press ‘shift-enter’ while cursor is in the code block

```
[3]: %jsroot on
#include "Rtypes.h"
#include "TH1.h"
#include "Math.h"
#include "Random.h"
#include <iostream>
using namespace std;

// ADJUST THESE NUMBERS AS PART OF THE EXERCISE
int Nsum=2;
int Nexp=10000;

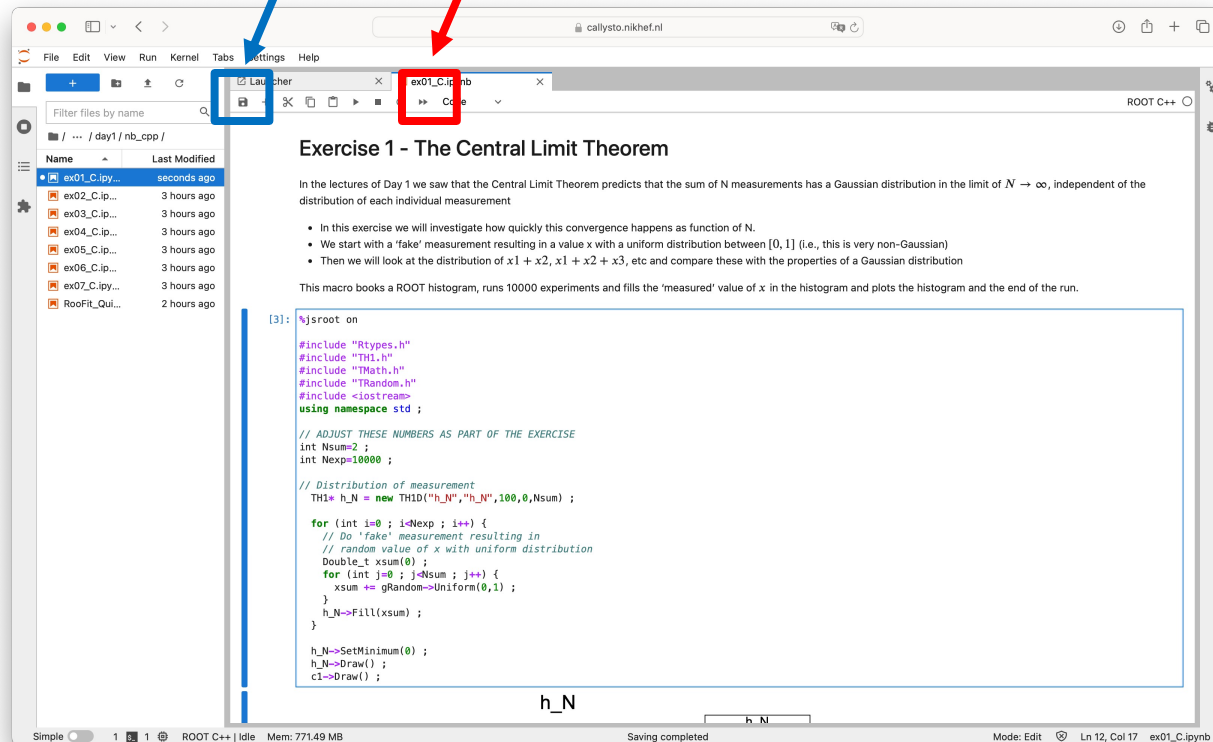
// Distribution of measurement
TH1* h_N = new TH1D("h_N","h_N",100,0,Nsum);

for (int i=0; i<Nexp; i++) {
    // Do 'fake' measurement resulting in
    // random value of x with uniform distribution
    Double_t xsum(0);
    for (int j=0; j<Nsum; j++) {
        xsum += gRandom->Uniform(0,1);
    }
    h_N->Fill(xsum);
}

h_N->SetMinimum(0);
h_N->Draw();
c1->Draw();
```

Running notebooks on callysto

- If you **save a notebook**, all your edits, and the latest outputs will be saved with it
- Stuck/problems? **Restart the kernel** and start compiling from scratch



Running notebooks on callysto

- Wondering what is happening?
 - [i] with $i=1, \dots, n$ before a code block means it has completed running
 - [*] before code block means 'running now'
 - [] before code block means 'not yet run'
- Also look at the **state of the kernel: Idle or Busy**

The screenshot shows a JupyterLab interface on a callysto.nikhef.nl server. The main window displays a notebook titled "Exercise 1 - The Central Limit Theorem". The code cell [3]: contains the following C++ code:

```
[3]: %jsroot on
#include "Rtypes.h"
#include "TH1.h"
#include "Math.h"
#include "TRandom.h"
#include <iostream>
using namespace std;

// ADJUST THESE NUMBERS AS PART OF THE EXERCISE
int Nsum=2;
int Nexp=10000;

// Distribution of measurement
TH1* h_N = new TH1D("h_N","h_N",100,0,Nsum);

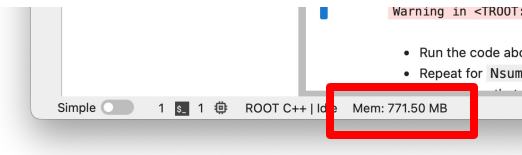
for (int i=0; i<Nexp; i++) {
    // Do 'fake' measurement resulting in
    // random value of x with uniform distribution
    Double_t xsum(0);
    for (int j=0; j<Nsum; j++) {
        xsum += gRandom->Uniform(0,1);
    }
    h_N->Fill(xsum);
}

h_N->SetMinimum(0);
h_N->Draw();
c1->Draw();
```

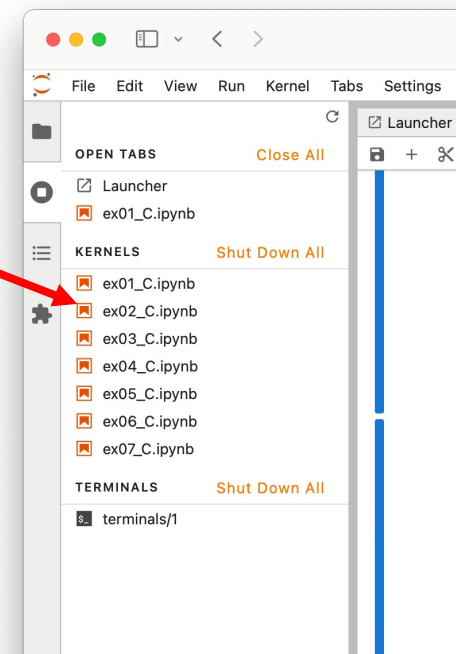
The kernel status bar at the bottom shows "ROOT C++ | idle" in a red box, indicating the kernel is not running.

Running notebooks on callysto

- **Please be a good citizen** – you are all running on the same machine!
- CPU will not be limiting factor, but *memory use can be with 40 users*



- If you have many kernels open, memory use can easily 5-10 Gb
- Every notebook opened starts a new kernel, but **closing the tabs does not close kernels**
- Click on the 2nd item of the menubar to see all open kernels.
- **If you are using >5 Gb of memory,** please clean up 'shut down' the kernels you are no longer using



Introduction to afternoon exercises

- There likely more exercises than you can do – that's a feature
- The provided order is the baseline recommendation for working, but if you have 'favorite' topics, feel free to start somewhere else
- *Don't spend too much time on a single exercise*
 - Each one start with a fairly concrete demonstration, and then ask you to investigate one or more features.
 - If you are not at the end of an exercises after 30 minutes (even with help), I suggest you move on the the next exercises anyway.
- *Don't hesitate to ask questions!* I am here for that purpose all day.
- I'll discuss solutions to a selection of exercises at the end of the afternoon

Today's exercises

ex1 – Central Limit Theorem

Demonstration of Central Limit Theorem ^ex01.C'

- ← 5000 numbers taken at random from a uniform distribution between [0,1].
 - Mean = $1/2$, Variance = $1/12$
- ← 5000 numbers, each the sum of 2 random numbers, i.e. $X = x_1 + x_2$.
 - Triangular shape
- ← Same for 3 numbers, $X = x_1 + x_2 + x_3$
- ← Same for 12 numbers, overlaid curve is exact Gaussian distribution

Important: tails of distribution converge very slowly CLT often *not* applicable for '5 sigma' discoveries

ex2 – Building & using models

Basics – Creating and plotting a Gaussian p.d.f ^ex02.C'

Setup gaussian PDF and plot

```

// Create an empty plot frame
RooPlot* xframe = w::x.frame() ;

// Plot model on frame
model.plotOn(xframe) ;

// Draw frame on canvas
xframe->Draw() ;
    
```

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

Unit normalization

Plot range taken from limits of x

ex3 – The Neyman-Pearson lemma

The Neyman-Pearson lemma ^ex03.C'

- Example of application of NP-lemma with two observables

- Cut-off value c controls type-I error rate ('size' = bkg rate)
Neyman-Pearson: LR cut gives best possible 'power' = signal eff.
- So why don't we *always* do this? (instead of training neural networks, boosted decision trees etc)

Wouter Verkerke, NIK-EF

ex4 – Two-dimensional models

Case study – dependence of 1-D model on another observable ^ex04.C'

- A common scenario for 2D modelling is the following: You observe that the mean reconstructed mass of some particle depends on another observable

Model for mass at $(y=0)$

$\text{sig}(m) = \text{Gaussian}(m, 92, 1)$

Model for mass at $(y=3)$

$\text{sig}(m) = \text{Gaussian}(m, 94, 1)$

↓

$\text{sig}(m, y) = \text{Gaussian}(m, \text{mean}(y), 1)$

Solution: introduce a function **mean(y)** that describes dependence of mean of y

Q: Is $\text{sig}(m, y)$ a proper 2-dimensional model?

Wouter Verkerke, NIK-EF

Today's exercises

ex5 – S+B models & low statistics

The power of building blocks – operator expressions ^ex05.C'

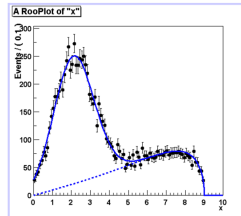
- Create a SUM expression to represent a sum of probability models

```
w.factory("Gaussian::gauss1(x[0,10],mean1[2],sigma[1])");
w.factory("Gaussian::gauss2(x,mean2[3],sigma)");
w.factory("ArgusBG::argus(x,k[-1],9.0)");

w.factory("SUM::sum(gfrac[0.5]*gauss1, g2frac[0.1]*gauss2, argus)");
```

- In composite model visualization components can be accessed by name

```
// Plot only argus components
w::sum_plotOn(frame,Components("argus"),
             LineStyle(kDashed));
```



ex6 – ML estimators

Parameter estimation – Maximum likelihood ^ex06.C'

- Practical estimation of maximum likelihood performed by minimizing the negative log-Likelihood

$$L(\vec{p}) = \prod_i f(\vec{x}_i; \vec{p})$$

↓

$$-\ln L(\vec{p}) = -\sum_i \ln F(\vec{x}_i; \vec{p})$$

- Advantage of log-Likelihood is that contributions from events can be summed, rather than multiplied (computationally easier)
- In practice, find point where derivative of $-\log L$ is zero

$$\left. \frac{d \ln L(\vec{p})}{d \vec{p}} \right|_{p_i = \hat{p}_i} = 0$$

- Standard notation for ML estimation of p is \hat{p}

Wouter Verkerke, UCSB