ROOT practicalities

- Practical workflow:
  - You can run on your laptop (if you installed it yourself), or run on stoomboot
  - You need ROOT version 5.34.21 or higher, preferably 5.34.36
  - I have tested all exercise macros with ROOT 5.34.36, which I installed on stoomboot. You can use this version as follows:

    ```
    source /project/atlas/user/verkerke/root-53436/bin/thisroot.sh
    (or .csh)
    ```
- Input files for exercises (ex1.C etc) can be found in directory `~verkerke/stat2016/` on stoomboot or login.nikhef.nl
- Annotations in exercises:
  - 'CODE' – means that you need to write some code
  - 'EXEC' – means that you need to run your code and interpret its output
- Macros have been tested with ROOT 5.34/21 & 5.34.36
  - Problems? Please ask (I didn't test everything on all platforms)

Exercise 1 – The central limit theorem

- In module 1 we saw that the Central Limit Theorem predicts that the sum of N measurements has a Gaussian distribution in the limit of N → ∞, 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
- Start with file ex1.C
  - 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.
  - EXEC: Look at the macro and run it ('`root -l ex1.C`' from the OS command line, or '`.x ex1.C`' from the ROOT command line)

Exercise 1 – The central limit theorem

- Modify the loop so that instead of filling the result of a single measurement in the histogram you store the result of `Nsum` measurements
  - CODE: Allocate a variable `xsum` that it is initialized to zero
    - The variable `Nsum` is already defined in the macro as first argument to macro ex1(). Its default value when unspecified is 1.
  - CODE: Make a loop from from j=1 to `Nsum` (inside the existing loop over i) and in new inner the loop add the value 'measurement' as returned by the '`gRandom`...' line to the value of `xsum`. Change the histogram filling code to use `xsum` instead of `x`
    - The histogram defined by the macro has its range already defined as [0,Nsum] so that the summed measurement values always fit in the range of the histogram
  - EXEC: Run the macro again now passing value 2 as argument for Nsum '`.x ex1.C(2)`' (or `root -l 'ex1.C(2)'` from the OS command line. Note that in this case the quotations are essential). Look at the distribution
  - EXEC: Repeat for `Nsum`=3,5,10,20 and 100.

Exercise 1 – The central limit theorem

- You will see that around `Nsum`=10 the distribution is already looks quite Gaussian.
  - This is however mostly for the 'core' of the distribution. The convergence of the tails of the distribution is much slower as we will see next in this exercise
- To compare the distribution to a Gaussian we compare the fraction of events in the range defined by 1,2,3,4,5 times the measured standard deviation (=√Variance) to the fractions expected for a Gaussian
  - I.e. we expect for a true Gaussian that 68% of the events is in the ±1 sigma range. Then we count which fraction of the xsum distribution is in that range
  - And we repeat for 2,3,4,5 sigma
- To do so we need to calculate (on paper) the expected standard deviation of the sum of N measurements with a uniform distribution.
  - Calculate first (on a piece of paper) the variance of a uniform distribution in the range [0,1].
  - To do so, use the formulas

$$\sigma \equiv \sqrt{V(x)} = \sqrt{\overline{x^2} - \overline{x}^2}$$

$$\overline{x} = \int x \cdot F(x)dx, \quad \overline{x^2} = \int x^2 \cdot F(x)dx$$

where F(x) is the distribution you are averaging over
(For this case F(x) is a uniform distribution in range [0,1])

Exercise 1 – The central limit theorem

- Then once you have variance for a single measurement of x, determine what the variance is for the sum of N identical measurements
  - If you need help, look at the slides on Central Limit Theorem of Lecture 1 (p44)
- Update the code to add this additional information
  - CODE: At the beginning allocate a variable `Nsigma1` and initialize it to zero. This will hold the number of events in the 'one-sigma' range.
  - CODE: In the 'experiment loop', once you have calculated `Xsum`, determine if the answer is inside or outside the 'one-sigma range', i.e. it is outside the range [-1*sigma,+1*sigma].
  - CODE: At the end of the loop print the fraction of events `Nsigma1/Ntot`, which is the fraction of events outside the one-sigma range of the distribution.
  - Compare it the fraction expected for a Gaussian distribution.
    Tip: You can get the exact fraction of events outside a n-sigma Gaussian distribution from the following ROOT expression:

    ```
    double  gaussfrac = TMath::Erfc(n/sqrt(2))
    ```

    where 'n' is the number of sigmas (i.e. 1 will give you 100%-68%≈32%)

Exercise 1 – The central limit theorem

- Note that there is a statistical uncertainty on the measurement of `Nsigma1/Ntot` which is (to good approximation) `sqrt(Nsigma1)/Ntot`
  - Compare Nsigma1, the stat. unc on Nsigma1, and the expected value of Nsigma1 for a Gaussian distribution on one line
  - EXEC: Do this for Nsum=2,5,10,20,100
  - You will see that 10000 experiments provides plenty precision to see that the one-sigma range of the distribution of Xsum converges rapidly to that expected for a Gaussian distribution.
- Now repeat the exercise for 2,3 sigma range.
  - CODE: To do so, add variables Nsigma2, Nsigma3, fill them in the event loop with the corresponding ranges and compare them (with their errors) to the matching fractions for a true Gaussian distribution.
  - EXEC: Do this for Nsum=2,5,10,20,100
  - Do you have enough statistics to measure the convergence for 2 and 3 sigma? If not, increase the number of experiments by e.g. a factory of 10
- Finally add the 4,5 sigma range
  - CODE & EXEC: How many experiments do you need to verify 5-sigma convergence?
    (Feel free to stop this exercise if runs start to take too long)

Exercise 1 – The central limit theorem

- What does it mean?
  - If you have done your exercise correctly you'll see the following results for the Nsum=20 run with Nexp=10.000.000 for 1,2,3,4,5 sigma

```
n = 3198780 frac = 0.319879  +/- 0.00017 Gauss = 0.317311   rel. = 0.008
n = 450384  frac = 0.0450384 +/- 6.7e-05 Gauss = 0.0455003  rel. = -0.010
n = 22954   frac = 0.0022954 +/- 1.5e-05 Gauss = 0.0026998  rel. = -0.149
n = 329     frac = 3.29e-05 +/- 1.8e-06  Gauss = 6.33425e-05 rel. = 0.480
n = 0       frac = 0         +/- 0       Gauss = 5.73303e-07
```

  - While the 2,3 sigma fractions are fairly close to Gaussian (rel=(frac-Gauss/Gauss)) the 4-sigma number is 50% off
  - E.g. your interpretation of how often a result 4 times the sqrt(variance) away from the central value happens is 50% off w.r.t the Gaussian distribution
  - Verifying 5 sigma results is a very time consuming business (even when a simulation of your measurement is as trivial as throwing a single random number)
- Conclusion: interpretation of large deviations expressed as 'N standard deviations' in terms of probabilities is difficult due to slow convergence of tails → To quantify a >3 sigma deviation, an explicit calculation is often needed

Wouter Verkerke, NIKHEF

Exercise 2 – Toy event generation

- This exercise demonstrates the principle of toy event generation through sampling (see slides Lecture 2 p9-12)
  - Copy input file `ex2.C` and look at it. The input file defines a nearly empty main function and a function '`double func(double x)`' is defined to return a Gaussian distribution in x.
  - The first step of this exercise is to sample `func()` to make a toy dataset. To do toy MC sampling we first need to know the maximum of the function. For now, we assume that we know that `func(x)` is a Gaussian and can determine the maximum by evaluating the function at `x=0`. Store the function value at `x=0` into a `double` named `fmax`.
  - Now write a loop that runs 1000 times. In the loop, you generate two random numbers: a `double x` in the range [-10,10] and a `double y` in the range [0,`fmax`]. The value of x is a potential element of the toy dataset you are generating. If you accept it, depends on the value of `y`. Think about what the acceptance criterium should be (consult the slides L2 p9-12 if necessary) and if it passes, store the value of `x` in the histogram.

Wouter Verkerke, NIKHEF

Exercise 2 – Toy event generation

- Allocate an integer counter to keep track of the number of accepted events. At the end of the macro draw the histogram and print the efficiency of the generation cycle, as calculated from the number of accepted events divided by the number of trial events
- Now change the code such that instead of doing 10000 trials, the loop will only stop after 10000 accepted events. Modify the code such that you can still calculate the efficiency after this change.
- Change the width of the Gaussian from 3.0 to 1.0. Run again and look at the generation efficiency. Now change it to 0.1 and observe the generation efficiency.

- Now we modify the toy generation macro so that it is usable on any function.
  - This means we can no longer rely on the assumption that the maximum of the function is at x=0.
  - The most common way to estimate the maximum of an unknown function is through random sampling. To that effect, add some code before the generation loop that samples the function at 100 random positions in `x` and saves the highest value found as `fmax`.

Exercise 2 – Toy event generation

- Change the width of the Gaussian back to 3.0 and run modified macro. Compare the `fmax` that was found through random sampling with the `fmax` obtained using the knowledge that the function was Gaussian (i.e `fmax=func(0)`).
- Now change the `func(x)` from the Gaussian function to the following expression:

```
(1+0.9*sin(sqrt(x*x)))/(fabs(x)+0.1)
```

  and verify that toy data generation still works fine.

- Finally we explore the limitations of sampling algorithms.
  - One runs generally into trouble if the empirical maximum finding algorithm does not find the true maximum. This is most likely to happen if you don't take enough samples or if the function is strongly peaked.
  - Choose the following `func(x)`

```
TMath::Gaus(x,0,0.1,kTRUE)+0.1;
```

    i.e. a narrow Gaussian plus a flat background and rerun the exercise
  - Now lower the number of trial samples for maximum finding to 10 and see what happens

Exercise 4 – Multi-Variate Analysis

- ROOT is distributed with the 'Toolkit for Multivariate Analysis', a toolkit that allows to train and apply many of the multi-variate analysis techniques shown in Lecture 3.
  - Here we will run it on a number of sample events
  - Copy input file `ex4_makesample.C`. This is a macro that can generate several 'toy' input samples.
  - Copy input file `ex4_driver_root532.C`.
    This is the driver macro to run the TMVA toolkit.

Exercise 4 – Multi-Variate Analysis

- Make sample #0
  - EXEC: Linux/MacOS: execute from the OS command line
    `'root -l -b  -q ex4_makesample.C(0)'`

    Windows: do `.x ex4_makesample.C(0)` from the ROOT command line)

- This make a file `sample0.root` which contains a sample of toy background events and toy signal events
  - Signal: Gaussian distribution in x (mean=-3, sigma=3)
  - Background: Gaussian distribution in x(mean=+3, sigma=3)
  - NB: There is a dummy (uniform) Y variable in the data because TMVA refuses to work with a single variable in some versions

Exercise 4 – Multi-Variate Analysis

- To analyze the first sample, start a ROOT session
  - EXEC: Execute '`.L ex4_driver_root532.C`', this loads the driver application
  - EXEC: Now analyze the first sample by issuing the following command on the ROOT prompt
    `ex4_driver_root532(0,"x,y","Fisher,BDT,MLP");`
  - This will train a Fisher discriminant, a Boosted Decision Tree and a Multi-Layer Perceptron for sample 0 using variables x,y
  - Observe how e.g. a BDT trains a lot quicker than a MLP, but takes longer to evaluate on the data.
  - While the training is running (~5 minutes), take a moment to review the techniques being trained in the slides of Lecture 3
  - When the training is finished, a window will pop up with various choices.

Exercise 4 – Multi-Variate Analysis

- Explore the following menu items in this order
  - 1a) Input distributions (just one for this example)
  - 4a) Classifier output distributions (observe characteristic spikiness of BDT output)
  - 4b) Same, but overlay of both test and training samples. Difference in these are indicative of overtraining (not in this sample)
  - 5b) ROC curve (signal vs background efficiency)
  - 5a) Efficiency curves (show signal and background efficiency vs discriminant, as well as S/sqrt(S+B) which helps to find optimal cut for a given amount of signal and background (Skip this one if you run on windows)

    Change the amounts of signal and background in the dialog box and see how the S/sqrt(S+B) changes shape
  - 9) – 11) Control plots for individual algorithms (these show e.g. network architecture, BDT structure etc...)
  - NB: If you run on Windows or don't have a compiler installed not all options will work (notable 5a will crash ROOT on windows w/o compiler, but some others may also not work.

Exercise 4 – Multi-Variate Analysis

- Now create sample 1
  - by running 'root -l -b -q ex4_makesample.C(1)'
  - Sample 1 has three observables :x,y,z
  - Signal = Gaussian(x,0,3)*Gaussian(y,0,3)*Gaussian(z,0,3)
  - Background = Flat in (x,y,z)
- Now analyze sample 1
  - Add the likelihood discriminant: execute

```
.x ex4_driver_root532.C(1,"x,y,z","Fisher,BDT,MLP,Likelihood")
```

  from the ROOT command line and look at the plots as for sample 0, but add the plots that the correlation (2a) and the plots that show the performance of decorrelation (2b, 2c,2d,1b,1c,1d)

- You will see that the performance of Fisher is very good for sample 0, but much worse for sample 1
  - Try to understand why that is (see slides on Fisher discriminant and MLP)

Exercise 4 – Multi-Variate Analysis

- Repeat the exercise for the other samples below
- Sample 2
  - Signal and background differ only in their correlation information
  - Signal = Gaussian({x,z,y},0,3) 80% correlation between (x,y) and 50% correlation between (y,z)
  - Background = Gaussian({x,z,y},0,3) 80% anti-correlation between (x,y) and 50% anti-correlation between (y,z)
- Sample 3
  - Multi-dimensional variant of sample 0
  - Signal = Gaussian(x,-3,3)*Gaussian(y,-3,3)*Gaussian(z,-3,3)
  - Background = Gaussian(x,+3,5)*Gaussian(y,+3,5)*Gaussian(z,-+3,5)
- Sample 4
  - Intertwined donuts. Two observables (x,y)
  - Signal = donut in (x,y) centered at (-2,-2), radius 5, width 1
  - Signal = donut in (x,y) centered at (+2,+2), radius 5, width 1
- If you have time left, you can edit ex4_makesample.cxx and code a few additional probability density functions of your choice
  - For syntax explanations see slides of Lecture 5