Topical Lectures Statistics – Exercises Set 1

Wouter Verkerke (Dec 2020)

General Instructions

Input files

Input files for exercises can be found in three places

1. At nikhef (stbc-i1.nikhef.nl) in directory `~verkerke/stats2020/`
2. At CERN (lxplus7.cern.ch) in directory `~verkerke/public/stats2020`
3. On the web at `http://www.nikhef.nl/~verkerke/stats2020`

Running ROOT

All exercises are based on ROOT. You are recommended to use version 6.14.04, in which all prepared material has been tested. To pick up a pre-installed ROOT version please execute the following setup script

```
source /cvmfs/sft.cern.ch/lcg/app/releases/ROOT/6.14.04/x86_64-
centos7-gcc48-opt/root/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-i1.nikhef.nl or stbc-i2.nikhef.nl only (these run CentOS7 – required for above ROOT version)

If you have an account at CERN, please work on lxplus7.cern.ch, this will select an CentOS7 node (required for above ROOT version)

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

Recommended Exercises for Monday Dec 7 - 2020

- **Ex1 – 'Warm up'**
- **Ex 2 – ML estimation of lifetime (includes a little of math on paper)**
- **Ex 3 – Biases in ML fits for low statistics (mostly 'try & run')**

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, p.d.f.s, 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",3,0.1,10) ;
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[val,min,max]`**.

- To create a pdf or function use **`ClassName::objectName(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","gauss",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 an custom syntax exists to simplify their use. The most important ones are shown here

- Addition of two (or more) pdfs

```
SUM::objName(frac1*pdf1,pdf2) ;         // shape-only pdf
SUM::objName(yield1*pdf1,yield2*pdf2) ; // extended pdf: N(exp) =y1+y2
```

- Multiplication of two (or more) pdfs

```
PROD::objName(pdf1,pdf2,…) ; // product of independent pdfs
PROD::objName(pdf1|x,pdf2,…) // product involving a conditional pdf
```

- Interpreted function expressions (all `TFormula` 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 dataset

```
RooDataSet* pdf.generate(observables,eventcount) ;
RooDataSet* pdf.generate(observables) ; for extended models only
```

- Unbinned Likelihood fit of model to data

```
pdf.fitTo(data) ;
RooFitResult* r = pdf.fitTo(data,Save()) ; // save extra info
```

- Plotting of data and model

```
RooPlot* frame = obs.frame() ; // creates empty plot frame
data.plotOn(frame) ; // plot data on frame
pdf.plotOn(frame) ; // project pdf on obs, normalize to data
```

Questions? Don't hesitate to ask!

Exercise 1 – An unbinned maximum likelihood fit

This a mostly a demonstration exercise only that shows some of the functionality and the syntax of the RooFit toolkit for data modeling that we'll be using in the next exercises

Copy file `ex01.C`, look at it and run it. This macro does the following

- It creates a Gaussian probability density function
- It generates an unbinned dataset with 10k events from that function
- It performs and unbinned ML fit of the Gaussian model to the toy dataset
- It makes a plot of the data and overlays it with the Gaussian model

Now comment out the part of the code labeled as '`block 1`' and run the macro again

- This code will print out the covariance matrix and correlation matrix of the fit parameters.

Now comment out the code labeled '`block 2`' and run again.

- This code will visualize the uncertainty of the model on the canvas using the error propagation technique. At 10K the event the uncertainty is very small (you can see it if you zoom in on the peak region of the pdf)
- Change the number of generated events from 10K to 100 and change the binning of the data in the plot from 100 bins to 10 bins (this is the argument in `the w.var("x")->frame()` call. Run again.
- Lower the number of generated events from 100 to 10 and run again. The error on the shape will now be significant, but you see that an unbinned ML can reliably fit very small event samples

Now comment out code block 3

- This will visualize the error on the p.d.f shape due to the uncertainty on the mean parameter only

Exercise 2 – Maximum Likelihood life time estimator

For certain simple cases it is possible to calculate the ML estimate of a parameter analytically rather than relying on a numeric estimate. A well-known case is that of the fit of a lifetime of an exponential distribution

Copy `ex02.C` and run it. This example performs an unbinned MLE fit to an exponential distribution of 100 events to estimate the lifetime.

Now we aim to construct the *analytical* ML estimator of the lifetime
(do this part on paper, not by computer)

- Write down the probability density function for the exponential lifetime distribution.
- It is essential that you formulate a *normalized* expression, i.e. Int F(t) dt = 1 when integrated from 0 to ∞.
  The easiest way to accomplish that is to divide whatever you expression you have by the integral of that expression over dt. *(You can calculate that normalization integral on paper)*
- Next write down the analytical form of the negative log-likelihood –log(L) for that probability density function given a dataset of N events labeled these x_i in your expression. *Be sure to also include the pdf normalization term in the expression*
- The analytical ML estimate of the lifetime tau then follows the requirement that d(-logL)/dtau = 0. Calculate the expression for this derivative solve and derive the value of tau for which the requirement d(-logL)/dtau holds.

Finally, implement the analytical calculation of MLE estimator for tau
in the code of `ex02.C`

- Uncomment block one, which implements a look over the dataset, retrieving the values of the decay time one by one and build your calculation of the analytical estimate of tau with that of the numeric calculation from `fitTo()`
- Explain why you might have minor discrepancies between the analytical and numeric calculations.
- Increase the event count from 100 to 10000 and run again

Exercise 3 – A signal+background background

The most popular statistical models used in particle physics describe the data as a sum of a signal and background component in some discriminating observables. In this exercise we will explore the features of such a model, especially at very low sample sizes.

Copy file `ex03.C`. This macro performs the following steps

- Construct a model that is the sum of Gaussian signal and an Exponential background
- Generate a toy dataset with 1000 events
- Perform an *unbinned* ML fit to the data
- Plot data and projection of the model

Run the macro and observe its behavior. Then try a few variants

- Reduce the sample size to 100 and 10 and try again. If needed you can adjust the binning for the data in the plot projection (this will not affect the fit since it use the data unbinned)
- Uncomment the bottom section that runs a MC Study of the properties of this model: it generates 1000 toy datasets, fits each of them and collects the fit statistics, by default on the `fsig` parameter. *Does the fit look unbiased?*
- Lower the sample size for the toys in the study from 1000 to 50, and then to 10 and run for each reduced size again. *Have your conclusions changed?*

One of the problems the fit runs into at low statistics is that the signal fraction `fsig` has a lower boundary at zero. If statistics are sufficiently small, ML fits will frequently run into this boundary, warping distributions.

- To mitigate boundary effects, decrease the lower bound on `fsig` from 0 to -1. Rerun the toy study for sample size 10. Does it improve? Rerun again for 100 to see how the behavior improves.

Finally, we push the model to see how well it can constrain `fsig` for data that doesn't contain any signal (*this strategy relates to understanding the potential for discovery of a signal – this will be discussed further in tomorrows lectures*).

- Change the default `fsig` value to zero, and run the study for sample size 100. Then decrease the sample size to 10 and run again.

As you, at sufficiently low sample size, ML estimators will suffer from biases. ML estimators are unbiased, per the ML theorem, *if such an unbiased estimator exists,* but for very low statistics samples such estimators will not exist. Generally, bias terms scale with 1/N, hence decreasing rapidly rising N also in comparison with the statistical error that scales with $1/\sqrt{N}$

Exercise 4 – A simple two-dimensional model

A simple extension of the 1-dimensional signal+background model of exercise 4 involves a variant where the mean of the Gaussian signal changes as function of a second observable y in the data

Copy file `ex04.C`. This macro performs the following steps

- Construct an extended model that is the sum of Gaussian signal and an Exponential background
- Generate a toy dataset with 1000 events
- Perform an *unbinned* ML fit to the data
- Plot data and projection of the model on the observable x

Run the macro and observe its behavior.

- Do you understand the shape of the signal in the plot?
- Change the slope in the `mean_func()` of the Gaussian signal model to a smaller number (or even zero) and see how the shape changes
- Does the precision of the estimate of `Nsig` depend on the magnitude of the slope of `mean_func`? Explain why (not)?
- Do you understand the (anti)correlation between the estimates of `Nsig` and `Nbkg`?
- You can (as an option) also visualize the model in 2 dimensions by uncommenting the last part of the macro. You may need to rotate the plot from its default viewing angle by drag-clicking on it.

Exercise 5 – Using the NP Lemma for event selection

The Neyman-Pearson lemma states that the optimal selection of selection of data points (x,y,z) that contain a mix background (hypothesis H0) and signal (hypothesis H1) is given by a contour of the likelihood ratio

$$L(H_0)/L(H_1) > c$$

where the critical value c can be chosen to obtain a selection with the desired tradeoff between purity (ratio of H_0 and H_1 in selection) and efficiency (fraction of H2 in selection). We are going to exploit this NP lemma to create a plot projection of a three-dimensional pdf on its observable x, while applying an NP-optimal selection criteria on its observables y,z.

Copy file `ex05.C`. This macro performs the following steps

- Construct a 3D Gaussian signal model, and a 3D Polynomial bkg model
- Generate a sample of 10.000 events from the model
- Fit the model to the data
- Plot a plain projection of the data and model on observable x

As is clear from the plot, when all data is projected in a single observable, there substantial background under the signal, as the projection on x does not take the discriminating information in observables y,z into account.

- Uncomment the return statement half-way the macro. The macro will now additionally do the following
    - Construct a NP-optimal selection criteria f $L(H_0)/L(H_1)$ where H_0=signal, and H_1=signal+background, for events using the (y,z) information of the model only
    - Make a plot of the data projected on x, only selecting those events that meet $L(H_0)/L(H_1) > 0.7$
    - Make a corresponding projection of the 3-model that integrates out only region selected by the requirement $L(H_0)/L(H_1) > 0.7$
- Observe the much-improved signal-to-background ratio in the new plot
- Try to change the Likelihood ratio cut value up and down and see what happens? *Why does $L(H_0)/L(H_1) > 0.8$ result in a empty plot?*

Finally we compare the sensitivity to signal significance for two potential workflows

- 1 – No event selection – big fit too all events
- 2 – NP-optimal event selection using observables y,z.
  Subsequent fit on selected events in observable x only

The first workflow is already executed in the macro – it is simply the initial fit. The second workflow can be enabled by removing the return statement close to the end of the macro. *How do does the signal significance differ?* (In this regime of signal yields you can approximate the significance as $Z=f_{sig}/\sigma(f_{sig})$. Repeat for several different settings of the Likelihood Ratio cut.

Exercise 6 – The important of accurate modeling

Statistical inference is only valid in the limit that the probability model that is used can accurately describe the data. While this seems trivially obvious, plenty of problems can arise in practice

Here we investigate a common area of problems – model with tail regimes that assign very low probabilities to events occurring in these regions

Copy file `ex06.C.` This macro performs the following steps

- Construct a narrow Gaussian probability model: the observable range spans about 20σ of the Gaussian.
- Fit sample of 100 points, drawn from the Gaussian to the model

Run the macro and observe its behavior. *Is the fit unbiased*?

- Uncomment the second code block. In this section, a single outlier event is added 'by hand' to the dataset and the fit is repeated. *How large is the fit bias due to this single event?*
- What happens if you move the outlier event to x=9? Can you explain why only the width parameter is biased and not the mean parameter?

Outlier events like the one manually added here can occur in real data if detectors or reconstruction techniques do not behave according their assumed specifications. To avoid strong biases due to (exceedingly) rare outlier points, it is prudent to include a term in models that can absorb such unexpected events.

- A prudent solution for narrow signals like the Gaussian model studied here is to add uniform background to the Gaussian signal that can model a small fraction of events that do not conform to the expected behavior
- Construct a model that is the sum of the original Gaussian plus a Uniform background model, where the fraction of Gaussian signal is floating and initialized to a value close to 1 (but not exactly one). Consult e.g. the model of `ex03.C,ex04.C` on the syntax to construct such model. Run the fit and visualize the result in the 3rd plot panel.
- What is the loss in precision on the estimate on the mean and sigma of the Gaussian model compared to the original fit without outlier and without outlier absorption term?

Exercise 7 – A likelihood fit for an efficiency curve

This exercise is mostly a demonstration. The goal of this exercise is to demonstrate that every modeling problem can be cast in the form of a probability model, so that any subsequent parameter estimation can be done with e likelihood fit.

Copy `ex07.C`. This macro presents a classical problem that is usually seen as 'X^2-fit' problem.

- A dataset has two observables: a continuous observable x, e.g., an energy of a triggered object, a label c that indicates if the event is accepted or rejected by the trigger.
- *The goal is to fit a turn-on curve of the efficiency of the trigger*. This efficiency function (which is clearly not a probability model itself!) has an S-curve in it that can be fit to describe a turn-on point in the data at some energy, and with some resolution (i.e. it is not a very sharp turn-on)
- The classical ('naïve') approach is to first construct an efficiency histogram from the data (fraction-accepted vs E). Then a X^2-fit is performed of the efficiency function f(E|p) with some parameters p to the observed efficiency histogram.
- Run the macro – and see the 'naïve' approach executed as a plain ROOT X^2-fit

Apart from the fact that the X^2-fit must be a binned fit, the highly non-Gaussian nature of the binomial errors on the efficiency data (especially close to 0 or 1) is cause for concern. It would thus be preferable to not do this. How can this parameter estimation problem be captured in a likelihood model?

- The first step is to realize that one needs to go back to the original 2-dimensional dataset D(E,c) where E is the energy observable and c is discrete labeling observable that can take states 'accept' and 'reject'
- Given an efficiency ε(E) that is valid at energy E, the probability model for the observable c is a Binomial model with n=2.
- The conditional probability model for this data is thus simply

  f(c|E) = Binomial(n=2,k=ε(E))

- The full pdf would then be f(c|E)*F(E), but this step can be omitted if there is no signal and background component to the full model as F(E) could simply be taken to be identical the observed distribution of E.

Remove the return statement in the code and run the likelihood fit alongside the ROOT X^2-fit.

- Observe that for 1000 events both fits are comparable in precision
- Lower the event count to 100 events, or even 50 events, rerun and observe the superior stability of the unbinned likelihood-based efficiency fit.

Exercise 8 – 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

Start with file `ex08.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.
- RUN: Look at the macro and run it ('`root -l ex8.C`' from the OS command line, or '`.x ex8.C`' from the ROOT command line)

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.

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

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

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%)

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)

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 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