Exercise/Hands-on #8

- Scientific Data Analysis Lab. course -

Experimenting Kolmogorov-Smirnov test for two samples

Alexis Pompili (UniBA)

Kolmogorov-Smirnov test - I

The Kolmogorov-Smirnov test is a method of the Statisticsl Analysis that allows to:

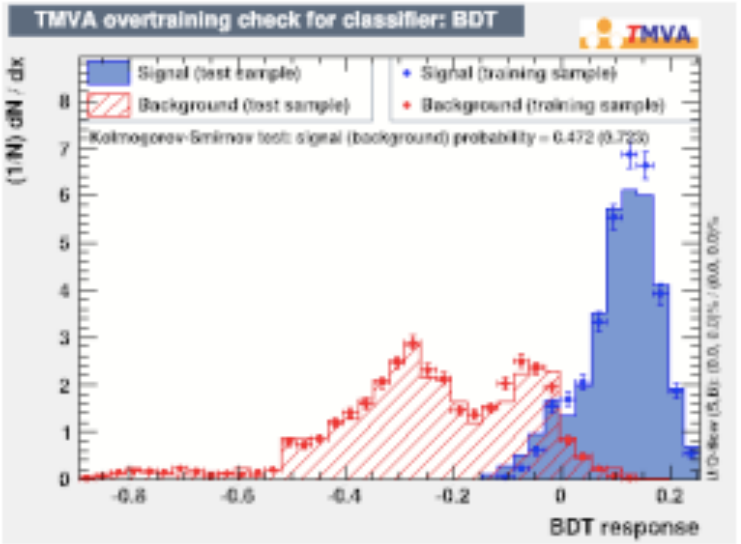
1) **compare a distribution of some observable for a data sample with a theoretical distribution** (that should describe the population of origin) with the aim … 1) to verify the hypothesis that the population from which the data are sampled is effectively the one under exam (for instance verify the normality of a variable), … 2) (in goodness-of-fit testing), to compare the data distribution with the model used in the fit [in a modified version though] (it is not used much in practise)

2) **compare the distributions of some observable for two data samples** with the aim ….

… **to verify that both data samples are coming from the same population**

(this is a common use of the test, as in the following example!)

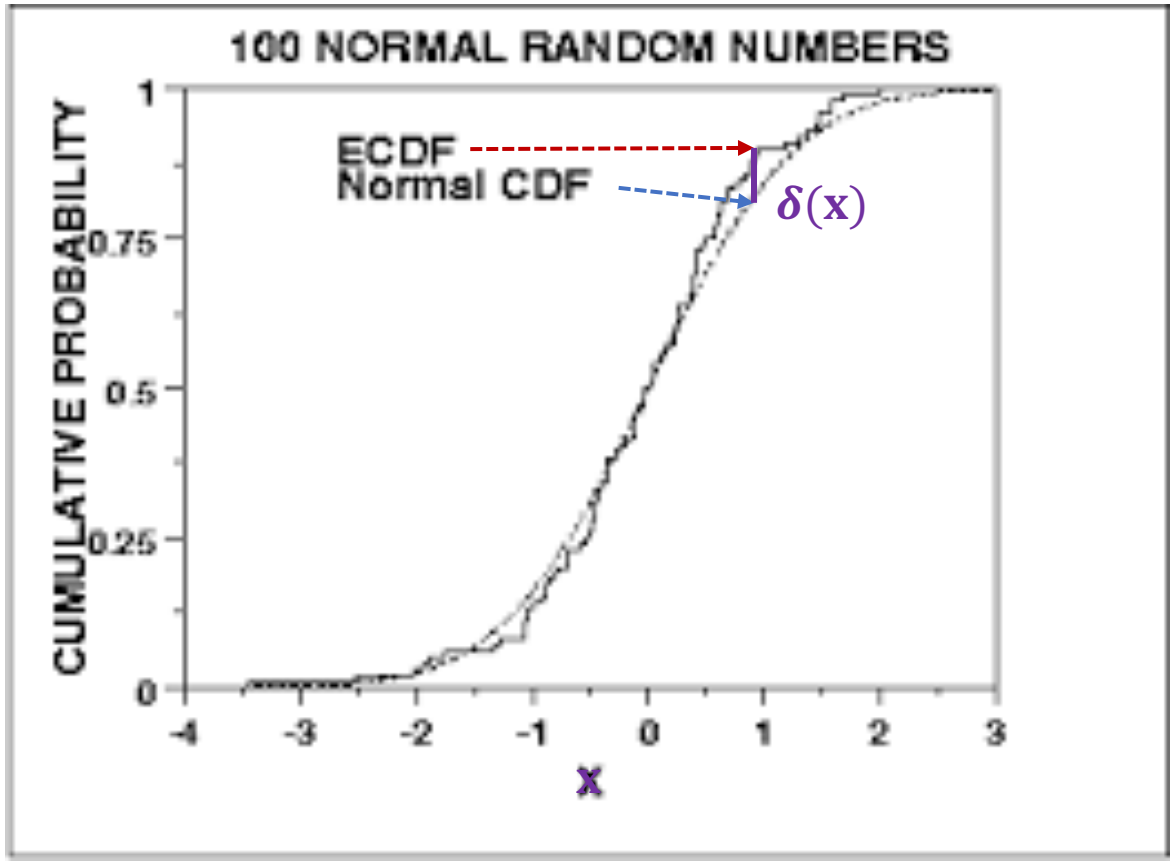
The plot on the right shows *training* and *testing samples* (for signal & background) for the output of a BDT classifier and shows the values of the Kolmogorov-Smirnov test for the compatibility of the two distributions for each category (sig or bkg), namely the compatibility of the training and testing classifier output histograms.
The KS test results represent how well the trained classifier describes the test sample.
If the values are low (because discrepancies between a pair of corresponding histograms are relevant) the classifier is considered *overtrained*, which means that it is biased for the particular training sample and does not describe describe the testing sample well.



(b) Classifier response histogram. The filled distributions are for the test sample and the dotted distributions are for the training sample.

Kolmogorov-Smirnov test - II

To be precise, the Kolmogorov-Smirnov test is a *non-parametric test* developed for two data samples by Smirnov in 1939, on the basis of Kolmogorov's considerations in 1933 concerning the comparison of one data sample with an underlying distribution (see the example below where … an ***Empirical CDF*** for a supposedly gaussian random variable is compared with a Gaussian CDF).



Kolmogorov considered the maximum absolute value of the deviation beyween the ECDF $F(x)$ and a CDF $\Phi(x)$:

$$\delta = max\{|F(x) - \Phi(x)|\} \equiv max\{\delta(\mathrm{x})\}$$

It can be demonstrated that if the hypothesis to be verified is true, then the probability to randomly obtain a value of δ not lower of a positive earlier chosen quantity δ_0 would be given by :

$$P(\delta \geq \delta_0) = F_{KS}(\delta'_0) \quad \text{with} \quad \begin{cases} F_{KS}(z) = 2\sum_{k=1}^{\infty}(-1)^{k-1}e^{-2k^2z^2} \\ \delta'_0 = \left(\sqrt{N} + 0.12 + \frac{0.11}{\sqrt{N}}\right)\delta_0 \end{cases}$$

The test is approximate but can be used already for small sizes of the samples ($N \geq 5$).

Kolmogorov-Smirnov test - warning if used for goodness-of-fit

The use of KS-test for goodness-of-fit purposes, i.e to compare the data distribution with the model used in the fit (parametric model with parameters estimated from the data sample itself) is believed to be wrong.

For instance check this link (https://asaip.psu.edu/articles/beware-the-kolmogorov-smirnov-test):

> **2. KS test probabilities are wrong if the model was derived from the dataset.** This constraint may seem strange. An astronomer often seeks a model that fits a chosen dataset, with best fit parameters chosen by least squares regression, maximum likelihood estimation, or Bayesian inference. It then seems natural to evaluate the goodness-of-fit of the model using the KS test. However, the theory underlying the KS and similar EDF-based tests require independence between the curves under consideration. The model must be derived from another dataset, or from external astrophysical considerations, for the standard KS test probabilities to be applied. Fortunately, the KS (or better, AD) statistic can still be computed, and the significance level of the difference between the EDF curves can be estimated by *bootstrap resamples* of the original dataset. Bootstrap resampling is conceptually and computationally simple, and the theory underlying the bootstrap guarantee that the resulting significance levels are unbiased for a wide range of situations.

Modified versions of the KS-test are possible to build a correct goodness-of-fit.

In HEP KS-test it is not much used in practise/literature for goodness-of-fit.

Exercise

We are going to use the KS-test in **ROOT** to compare two samples derived (by means of **RooFit**) of an identical Gaussian PDF.

The prepared simple macro **KS_test.C** will be executed as follows:

```
////////////////////////////////////////////////////////////////////////////////
//
// root [0] .L KS_test.C
// root [1]  myKStest(1000000, 1000, 1010, 400)
//
//--- providing #events, seed1#, seed#2, #bins
//
////////////////////////////////////////////////////////////////////////////////
```

We'll go throughout the macro code in next slides.

```
#include "RooPolynomial.h"
#include "RooRealVar.h"
#include "RooBreitWigner.h"
#include "RooNumConvPdf.h"
#include "RooVoigtian.h"
#include "RooGaussian.h"
#include "RooExponential.h"
#include "RooDataSet.h"
#include "RooDataHist.h"
#include "RooAbsData.h"
#include "RooMinuit.h"
#include "RooPlot.h"
#include "RooChebychev.h"
#include "RooAddPdf.h"
#include "RooArgList.h"
#include "TH1F.h"
#include <vector>
#include "TCanvas.h"
//
#include "RooRandom.h" // needed for Randomizer
//
using namespace RooFit; //----Working in RooFit//
//
////////////////////////////////////////////////////////////////////////////////
//
// root [0] .L KS_test.C
// root [1]  myKStest(1000000, 1000, 1010, 400)
//
//--- providing #events, seed#1, seed#2, #bins
//
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
//
void myKStest(int events=1000000, int seed1=1000, int seed2=1010, int bins=400) {
  //
  //-- Remember: external values override those dummy initializing values!
  //
  RooRealVar xvar("xvar", "", -12, 12);
  xvar.setBins(bins);
  //
  // Gaussian-1
  RooRealVar mean1("m1", "mean1", 0.5, -0.5, 1.5);
  RooRealVar sigma1("#sigma1", "sigma1", 1.8, 0.1, 5.);
  RooGaussian Gauss1("G1", "Gauss1", xvar, mean1, sigma1);
  //
  //---Gaussian-2 (chosen same as the 1st)
  RooRealVar mean2("m2","mean2", 0.5, -0.5, 1.5);
  RooRealVar sigma2("#sigma2", "sigma2", 1.8, 0.1, 5.);
  RooGaussian Gauss2("G2", "Gauss2", xvar, mean2, sigma2);
  //
  //-- note that alternatively you can try a Voigtian
  //
  RooRealVar meanV("meanV", "meanV", 0.5, -0.5, 1.5);
  RooRealVar gammaV("gammaV", "gammaV", 1.8, 0.1, 5.);
  RooRealVar sigmaV("sigmaV", "sigmaV", 1.1, 0.1, 5.);
  RooVoigtian Voigt("Voigt","Voigtian", xvar, meanV, gammaV, sigmaV);
  //
  ////////////////////////////////////////////////////////////////////////////////
```

PDF-1 & PDF-2 are identical here

(of course you can try a different PDF-2)

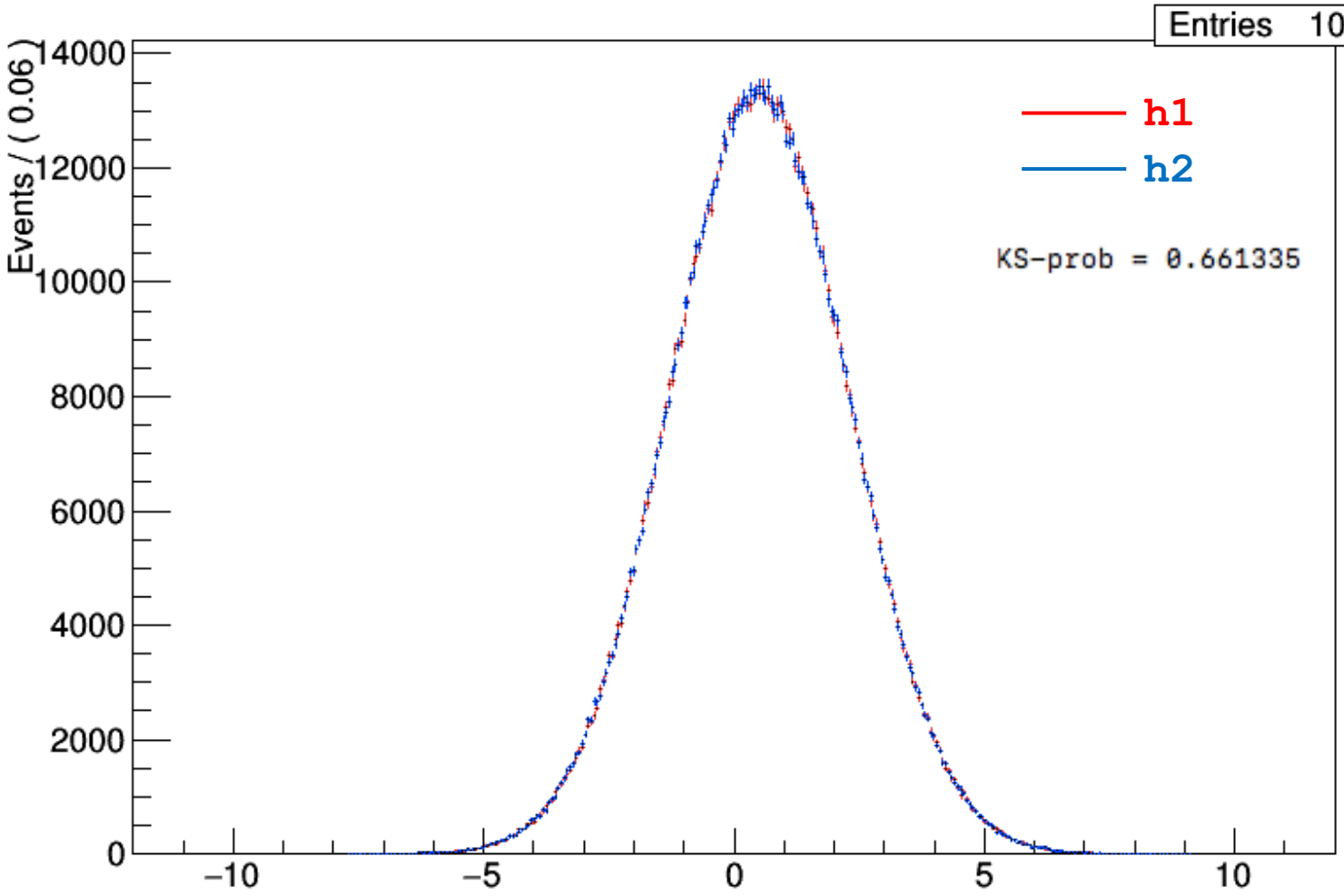
```
////////////////////////////////////////////////////////////////////////////
// Generating data : here we want to compare the samples of the same Gaussian PDF
////////////////////////////////////////////////////////////////////////////////////
//
cout << "\n------Generating " << events << " events" << " for PDF-1\n" << endl ;
cout << "\n------Remember: initial values for fitting step are the generated (true) values in generation ---" << endl;
//
// -- PDF-1
//
RooRandom::randomGenerator()->SetSeed(seed1);
RooDataSet* data1 = Gauss1.generate(xvar,events);
//
cout << "\n------Generating " << events << " events" << " for PDF-2\n" << endl ;
cout << "\n-----Remember to change seed every time to get different distributions------" << endl;
//
// -- PDF-2 (you can choose a Crystal Ball as an exercise to check the tail effect)
//
RooRandom::randomGenerator()->SetSeed(seed2);
RooDataSet* data2 = Gauss2.generate(xvar,events);
//RooDataSet* data2 = Voigt.generate(xvar,events); // to compare to a Voigtian
//
////////////////////////////////////////////////////////////////////////////
//
// The method createHistogram of the class RooDataSet allows to get a TH1 from a RooDataSet:
//
TH1 *h1 = data1->createHistogram("h1",xvar,Binning(bins));
TH1 *h2 = data2->createHistogram("h2",xvar,Binning(bins));
//
//--------------------------------
//
TCanvas *myC = new TCanvas("RooCanvas","Roofit Canvas", 850, 600);
//
myC->cd();
gStyle->SetOptStat("e");
h1->SetLineColor(2);
h1->GetYaxis()->SetTitleOffset(1.29);
h1->SetTitle("");
h1->Draw();
//
h2->SetLineColor(4);
h2->GetYaxis()->SetTitleOffset(1.29);
h2->SetTitle("");
h2->Draw("Same");
//
myC->SaveAs("two-histograms-two-gaussians.png");
//
Double_t ks = h1->KolmogorovTest(h2);  // KolmogorovTest is a method of class TH1
//
//write KS-test probability value at screen:
cout << "\n KS-prob = " << ks << endl;
//
////////////////////////////////////////
//
if (myC)
  {
   myC->Close();
   delete myC;
  }
}/
```

Sample 1 generated from PDF-1 with seed1

Sample 2 generated from PDF-2 with seed2

Running the code several times (changing everytime generation seeds) provides a KS probability value mostly between 0.5 and 1.

The result is:



(seed1=1000)
(seed2=1010)

Notes: **If you run with seed1=seed2 … the KS-probability will be 1 (as expected)!**

If you try the Voigtian (already implemented) vs Gaussian (even trying to adjust widths), the KS-probabilities will be 0

Homework

Try a Crystal ball against a Gaussian (they have to share mean and sigma and thus differ just by the tail).
To get non null probabilities try a very small tail.

Note: since histograms are normalized by generating the same number of events,
generate the Crystal Ball with a number of events slightly larger than the Gaussian
(about the number of events in the tail) so that the core Gaussian has the same area of the reference Gaussian

Note: according to the material at this link (https://wwwcdf.pd.infn.it/labo/twoup7.pdf) …
the KS-test is very sensible to differences in the central regions of data (around the mean value of the distributions)
[this is something that we will experience in the Part II (see next slides)], while it is not very effective in discriminating between two distribution that differ significantly only in their tails.
After executing the homework above - in the classroom - we found that it is not really true by comparing a Crystal Ball with a Gaussian (the former sharing the same parameters of its core Gaussian with the latter) and trying very small tails.

Further exercise - PART II

We are going to use the KS-test in **ROOT** to compare two samples derived (by means of **RooFit**) of an identical Gaussian PDF.
We now generate - in a loop - 200 pairs of samples from the Gaussian and we plot the histogram of the KS values.
We also want to investigate the dependency of this distribution from binning effects.

The prepared simple macro **KS_test_loop.C** will be executed as follows:

```
////////////////////////////////////////////////////////////////////////////////
//
// root [0] .L KS_test_loop.C
//
// root [1]  myKStest(1000000, 1000, 2000, 400)
//
//--- providing #events, seed#1, seed#2, #bins
//
////////////////////////////////////////////////////////////////////////////////
```

We'll go throughout the macro code in next slides.

```
#include "RooPolynomial.h"
#include "RooRealVar.h"
#include "RooBreitWigner.h"
#include "RooNumConvPdf.h"
#include "RooVoigtian.h"
#include "RooGaussian.h"
#include "RooExponential.h"
#include "RooDataSet.h"
#include "RooDataHist.h"
#include "RooAbsData.h"
#include "RooMinuit.h"
#include "RooPlot.h"
#include "RooChebychev.h"
#include "RooAddPdf.h"
#include "RooArgList.h"
#include "TH1F.h"
#include <vector>
#include "TCanvas.h"
//
#include "RooRandom.h" // needed for Randomizer
//
using namespace RooFit; //----Working in RooFit//
//
////////////////////////////////////////////////////////////////////////////////
//
// root [0] .L KS_test_loop.C
//
// root [1]  myKStest(1000000, 1000, 2000, 400)
//
//--- providing #events, seed#1, seed#2, #bins
//
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////
//
void myKStest(int events=1000000, int seed1=1000, int seed2=1010, int bins=400) {
  //
  //-- Remember: external values override those dummy initializing values!
  //
  RooRealVar xvar("xvar", "", -12, 12);
  xvar.setBins(bins);
  //
  // Gaussian-1
  RooRealVar mean1("m1", "mean1", 0.5);
  RooRealVar sigma1("#sigma1", "sigma1", 1.8);
  RooGaussian Gauss1("G1", "Gauss1", xvar, mean1, sigma1);
  //
  //---Gaussian-2 (chosen same as the 1st)
  RooRealVar mean2("m2","mean2", 0.5);
  RooRealVar sigma2("#sigma2", "sigma2", 1.8);
  RooGaussian Gauss2("G2", "Gauss2", xvar, mean2, sigma2);
  //
```

```
////////////////////////////////////////////////////////////////////////////////
// Generating data : here we want to compare the samples of the same Gaussian PDF
////////////////////////////////////////////////////////////////////////////////
//
int seed1_loop, seed2_loop;
Double_t ks;
//
double vecKS[200];
TH1D *histKS = new TH1D("histKS","KS values 1M 400bins", 20, 0., 1.);
//
for (Int_t j=0; j<200; j++)  // we choose to measure the KS-probability 20 times
   {
    //
    // -- PDF-1
    //
    seed1_loop = seed1+(j*5);
    RooRandom::randomGenerator()->SetSeed(seed1_loop);
    RooDataSet* data1 = Gauss1.generate(xvar,events);
    //
    // -- PDF-2
    //
    seed2_loop = seed2+(j*5);
    RooRandom::randomGenerator()->SetSeed(seed2_loop);
    RooDataSet* data2 = Gauss2.generate(xvar,events);
    //
    TH1 *h1 = data1->createHistogram("h1",xvar,Binning(bins));
    TH1 *h2 = data2->createHistogram("h2",xvar,Binning(bins));
    //
    ks = h1->KolmogorovTest(h2);
    vecKS[j] = ks;
    cout << "\n seed1 = " << seed1_loop << " and seed2 = " << seed2_loop << " give KS-prob = " << vecKS[j] << endl;  // to check all is working correctly
    //
    delete h1;
    delete h2;
    //
    histKS->Fill(vecKS[j]);
    //
   }
//
//
//------------------------------
//
TCanvas *myC = new TCanvas("RooCanvas","Roofit Canvas", 850, 600);
//
myC->cd();
histKS->SetMaximum(60);
histKS->SetMinimum(0.);
gStyle->SetOptStat(1110);
histKS->Draw();
myC->SaveAs("KS-test_values_distribution_1Mev_400bins.png");
//
//
////////////////////////////////////////
//
if (myC)
  {
   myC->Close();
   delete myC;
  }
```

initialization before loop

seed1 goes from 1000 to 1995

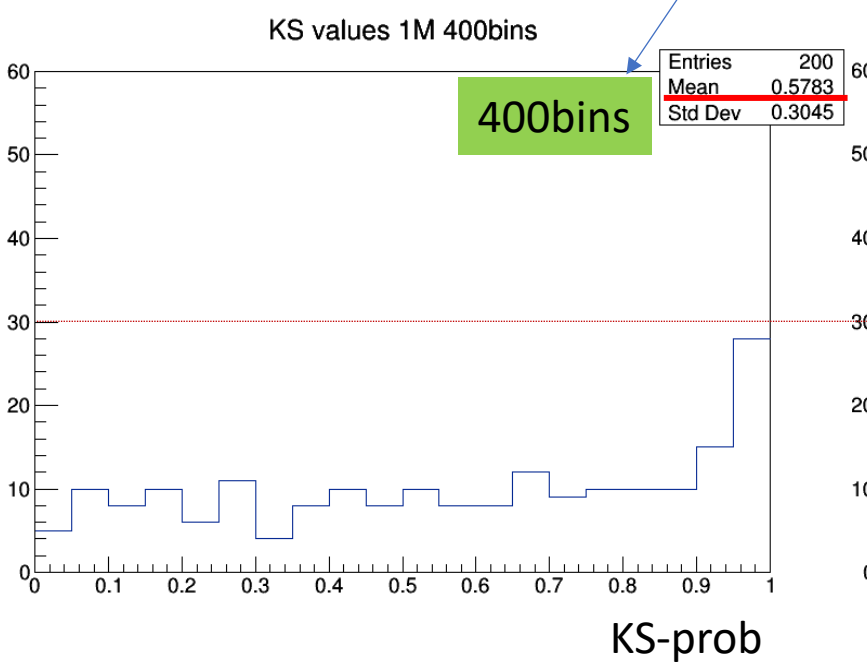
seed2 goes from 2000 to 2995

for loop

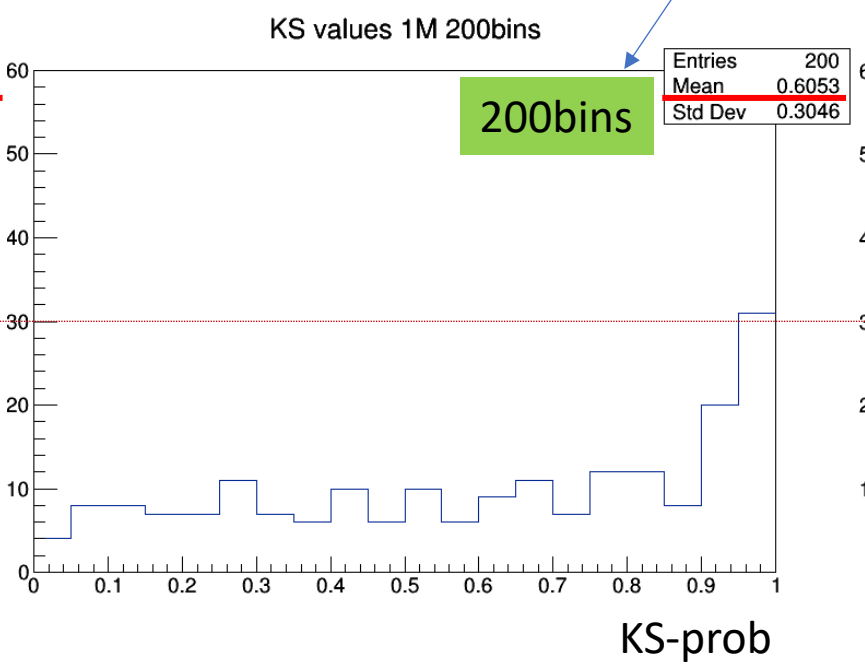
filling the histograms with KS-values

Using this macro we get:

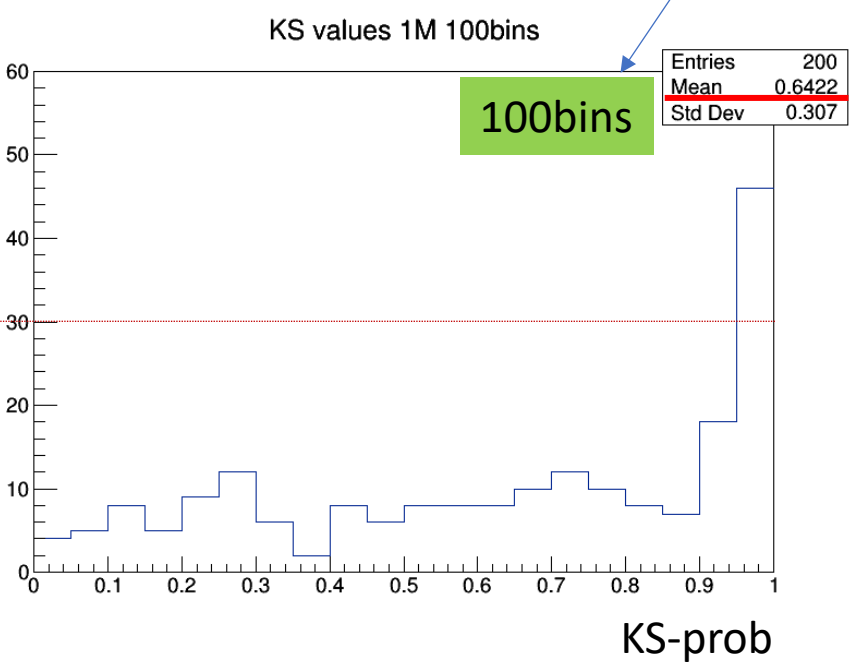
```
root [0] .L KS_test_loop.C
root [1] myKStest(1000000, 1000, 2000, 400)
```




```
root [0] .L KS_test_loop.C
root [1] myKStest(1000000, 1000, 2000, 200)
```




```
root [0] .L KS_test_loop.C
root [1] myKStest(1000000, 1000, 2000, 100)
```



For the same 1M generated events, reducing the number of bins seems to provide more robustness (less prone to fluctuations)