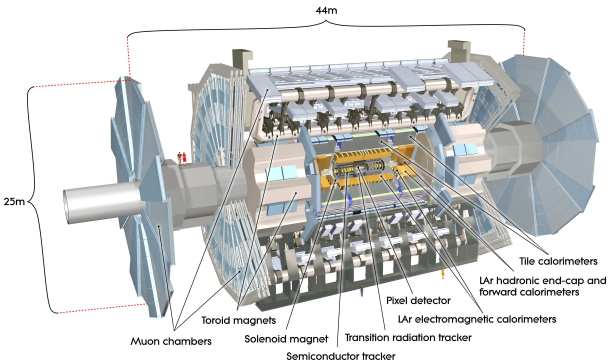
The ROOT framework
– Lecture and Tutorial –

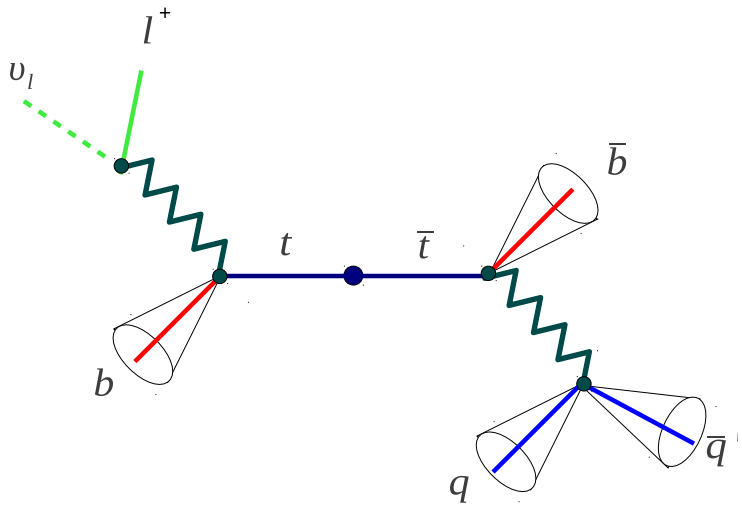
Andrea Knue
HASCO Summer School
Goettingen, July 2015

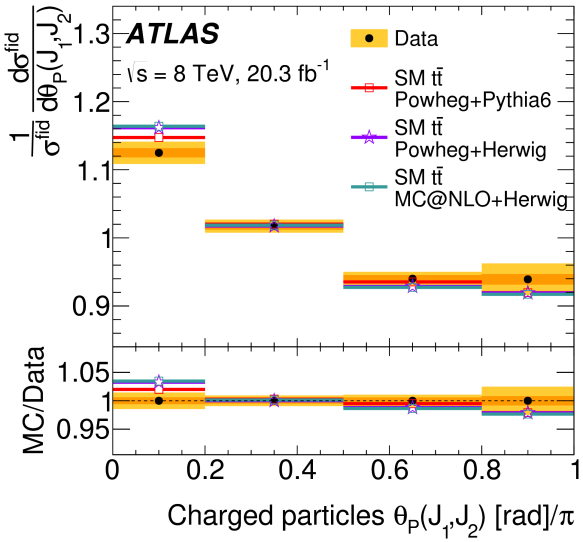
University of Glasgow

Our job in a (very) simplefied way

– HEP Experiment – + **– Process of interest –**







= fancy plot?

http://arxiv.org/abs/1506.05629

Not that simple!

Short Brainstorming:

1. What information do we have?
2. What information are we interested in?
3. How do we get there?

What do you need for data analysis?

A lot of important steps before extracting the final quantity:

- experiment $\rightarrow$ data taking!
- detector calibration
- event selection
- but also: produce simulation (Monte Carlo generators) to compare with!
- run all steps over data and MC
- write out ntuple for actual analysis

What tools do we need (basic view)

- plot histograms and measured values
- fit distributions
- add histograms
- stack histograms and compare to data
- make ratio plots, compare shapes
- have proper labelling
- extract values from data

➔ all this (and more) can be done with ROOT

Disclaimer

- info from https://root.cern.ch/
- took a bit of orientation also from last year ▸ Link
- some examples shown today: from $ROOTSYS/tutorials/

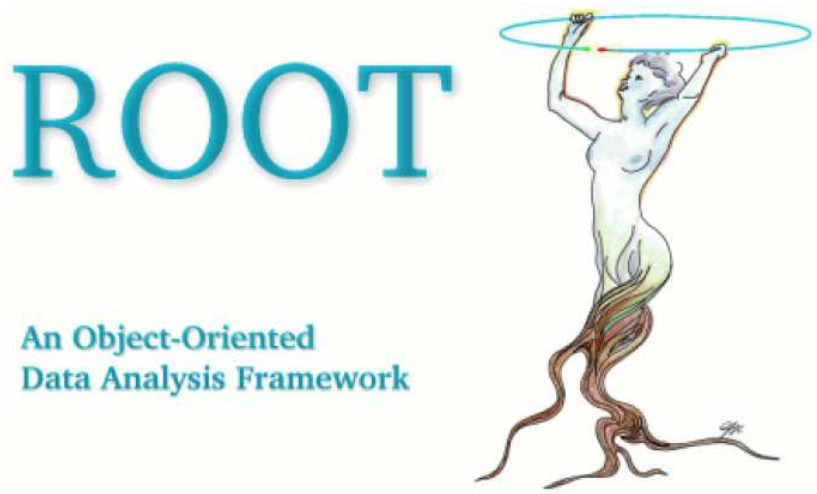
Introduction: What is ROOT?

- framework for data analysis (started 1994)

  ↪ before: PAW, Fortran based

  ↪ now: object oriented, based on C++

- developed for High Energy Physics
- new features implemented constantly
- find latest version for download here



Introduction: What is ROOT II

- **simple data analysis:** read data, fill histogram, handle four-vectors
- make fits, write out results
- make fancy plots for your thesis, paper etc
- includes also more **advanced statistical methods:**
  - ↪ Nuisance parameter fits: RooFit
  - ↪ Unfolding: RooUnfold
  - ↪ Multivariate Analyses: TMVA

Why object-oriented programming language?

- classes allow modular structure, decreased level of complexity
- allows classes to inherit from other classes

  $\hookrightarrow$ makes development much easier

  $\hookrightarrow$ object can easily be extended
- allows abstraction and polymorphism

How to get ROOT?

a) If you are working at CERN (lxplus)

ssh -Y username@lxplus.cern.ch

source /afs/cern.ch/sw/lcg/contrib/gcc/4.8/x86_64-slc6/setup.sh

source /afs/cern.ch/sw/lcg/app/releases/ROOT/6.02.05/x86_64-slc6-gcc48-opt/root/bin/thisroot.sh

b) on your laptop

- to install locally ▸ Download here!
- very easy to install on Linux
- works on Windows or MacOS too ▸ Link to versions

First steps

If root is properly installed on your computer, just type: root

```
-bash-4.1$
-bash-4.1$ root
  ****************************************
  *                                      *
  *        W E L C O M E  to  R O O T     *
  *                                      *
  *   Version   5.34/10    29 August 2013  *
  *                                      *
  *  You are welcome to visit our Web site *
  *          http://root.cern.ch         *
  *                                      *
  ****************************************

ROOT 5.34/10 (heads/v5-34-00-patches@v5-34-10-5-g0e8bac8, Sep 04 2013, 11:52:19 on linuxx8664gcc)

CINT/ROOT C/C++ Interpreter version 5.18.00, July 2, 2010
Type ? for help. Commands must be C++ statements.
Enclose multiple statements between { }.
root [0]
```

➔ if you want to avoid the logo to open, just type root -l instead of root .

Random generators

➔ computers: not truly random but **pseudo random**

➔ good enough for a lot of applications

Where do we need them?

- emulate detector conditions: smear simulated data (Monte Carlo) to model resolution of detector
- apply object calibrations (electron, muon jets) to match MC to data

What do we need from them?

- long period $\rightarrow$ should not repeat themselves too quickly
- fast algorithm: $\rightarrow$ have to do this quite often!

Random generators: how random is random?

ROOT comes with several random generators:

1. TRandom: periodicity: 10^9, 34 ns/call (no NOT use!)
2. TRandom1: periodicity: 10^{14}, 242 ns/call
3. TRandom2: periodicity: 10^{26}, 37 ns/call
4. TRandom3: **default, use only this one!**
   - $\hookrightarrow$ Mersenne-Twister generator
   - $\hookrightarrow$ very long periodicity: 10^{600}
   - $\hookrightarrow$ resonably fast: 42 ns/call

Random generators II

Choice of seed

- need to choose starting point (seed)
- choose specific value (for example 1234)

  $\hookrightarrow$ then the simulation can be reproduced (code debugging)
- choose seed=0:

  $\hookrightarrow$ take system time of the computer,

Examples for functions in TRandom3

- Uniform(x1)

  → calls random number uniformly distributed between 0 and x1

- Gauss(mean, sigma)

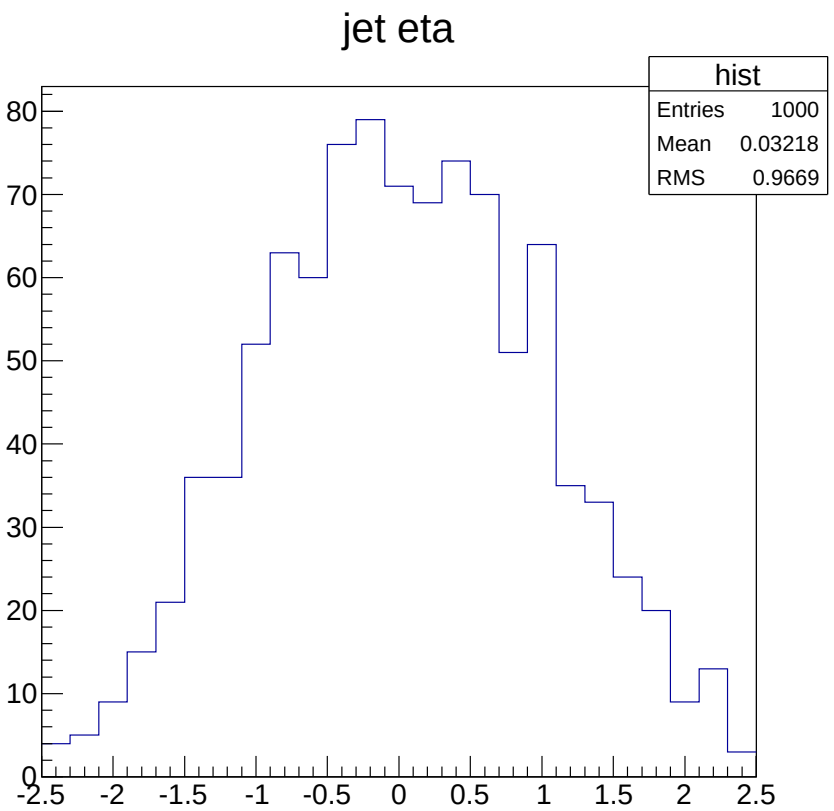
  → calls random number in gaussian distribution around mean with width sigma

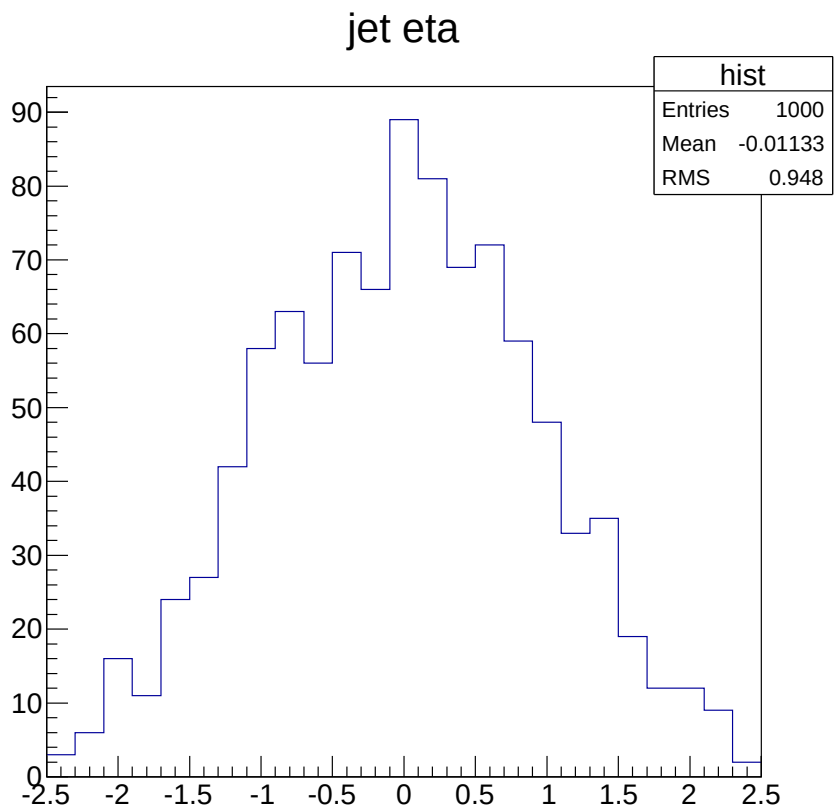
- exp(tau)

  → calls random number in exp distribution with exp(-t/tau)

- also have landau, poisson, binomial...

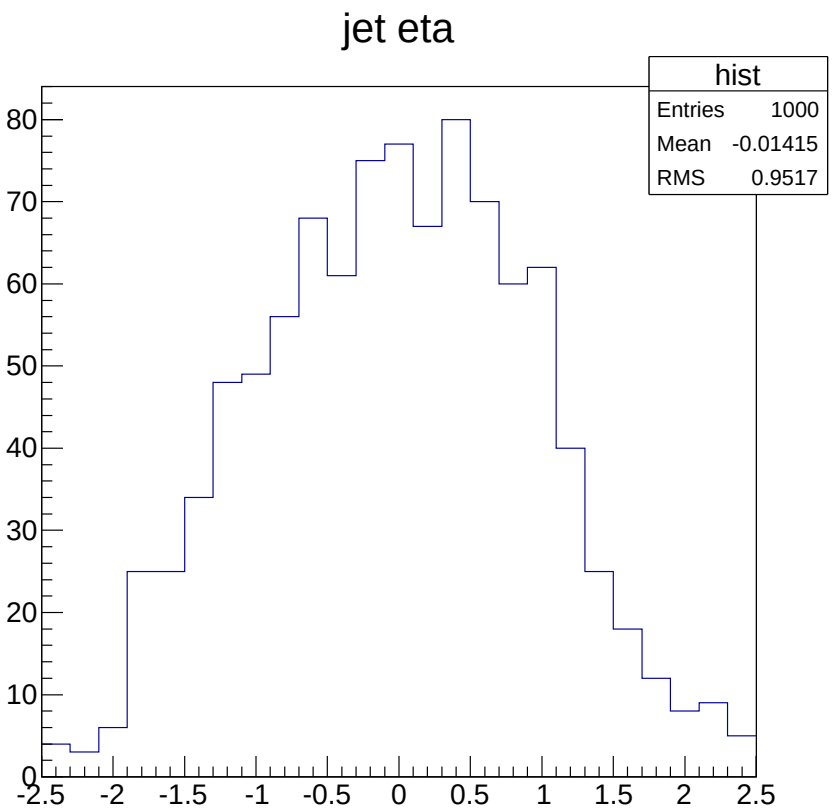
Two distributions: gaussians with seed 0

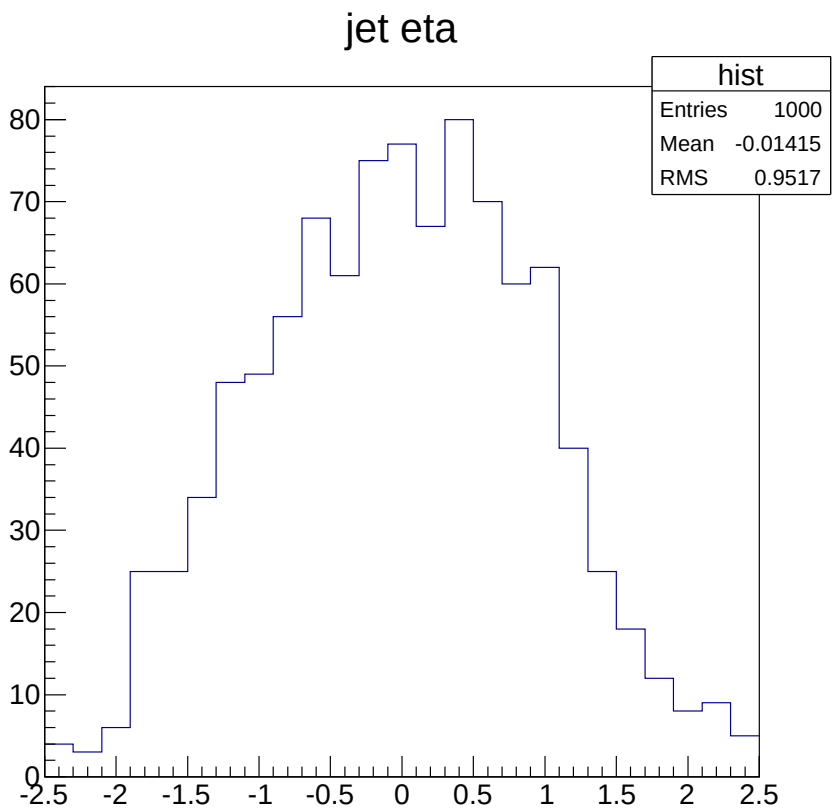




➔ run same code twice with seed 0: different distributions

Two distributions: gaussians with seed 1234

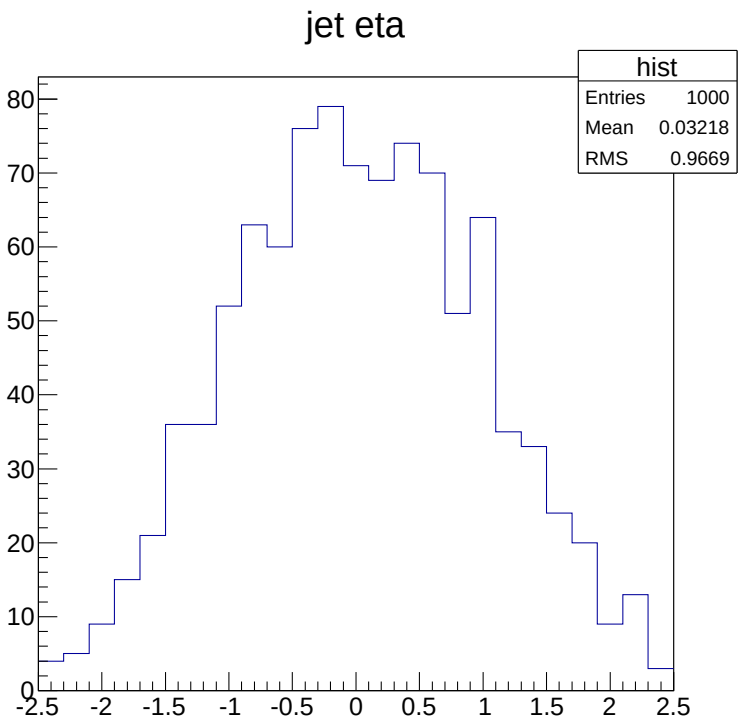




➔ run same code twice with seed 1234: same distributions !

How do we get this simple plot?

```
int TestCode(){

  // define canvas for output plot
  TCanvas  *c1     = new TCanvas("c1","The FillRandom example", 200, 10, 700, 700);

  // define 1D histogram with 25 bins and range -2.5 to 2.5
  TH1D     *fHisto = new TH1D("hist", "jet eta", 25, -2.5, 2.5);

  // initialize random generator
  TRandom3 *fRand  = new TRandom3();

  // seed seed: take system time
  fRand -> SetSeed(0);

  int stats1 = 1000;

  // fill histogram with 1000 random values
  for(int i = 0; i < stats1; ++i){

    double value = fRand -> Gaus(0.0, 1.0);

    fHisto -> Fill(value);

  }

  // draw histogram to canvas
  fHisto -> Draw();

  // print canvas
  c1     -> Print();

}
```



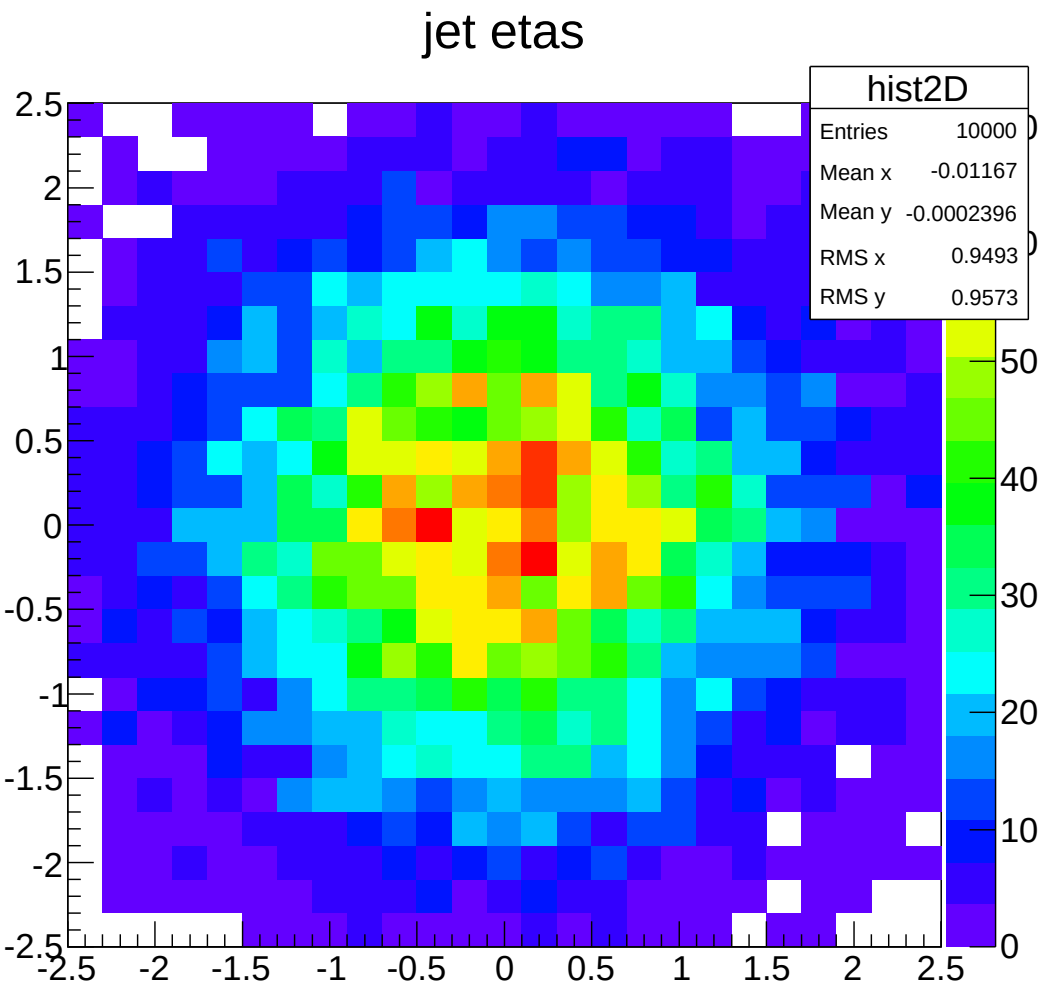
➔ write this into file TestCode.C

➔ now call in command line: root TestCode.C

Example code for 2D plot

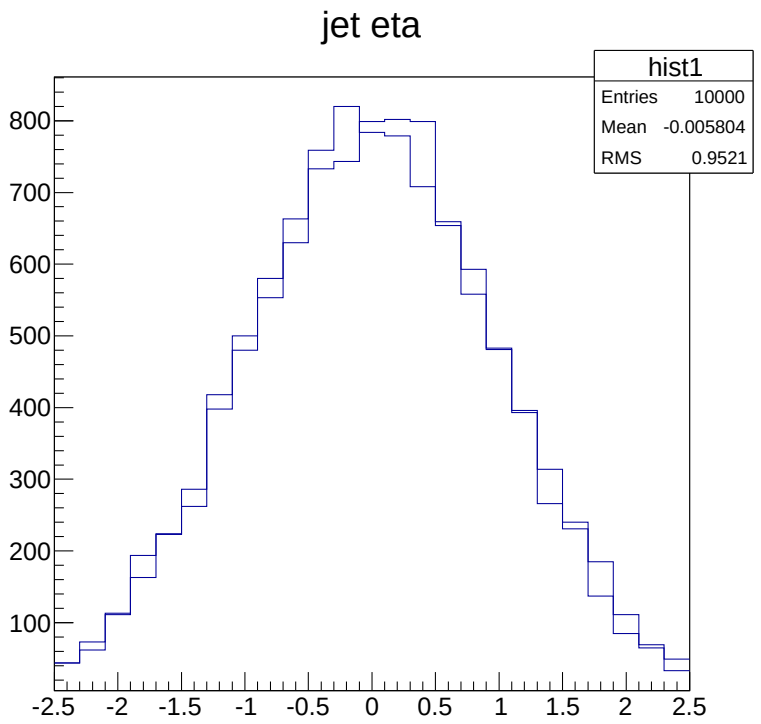
```
int TestCode2D(){

  // define canvas for output plot
  TCanvas  *c1      = new TCanvas("c1","The FillRandom example", 200, 10, 700, 700);

  // define 2D histogram with 25 bins and range -2.5 to 2.5
  TH2D     *fHisto1 = new TH2D("hist2D", "jet etas", 25, -2.5, 2.5, 25, -2.5, 2.5);

  // initialize random generator
  TRandom3 *fRand   = new TRandom3();

  // seed seed to system time: 0
  fRand -> SetSeed(0);

  int stats1 = 10000;

  // fill histogram with 10000 random values
  for(int i = 0; i < stats1; ++i){

    double value1 = fRand -> Gaus(0.0, 1.0);
    double value2 = fRand -> Gaus(0.0, 1.0);

    fHisto1 -> Fill(value1, value2);

  }

  // draw 2D plot with colour scheme
  fHisto1 -> Draw("COLZ");

  // print canvas
  c1     -> Print();
  c1     -> Print("test2D.root"); // write plot to root file for later usage

}
```

Plot output: no correlation



Now put 2 Histograms in one plot

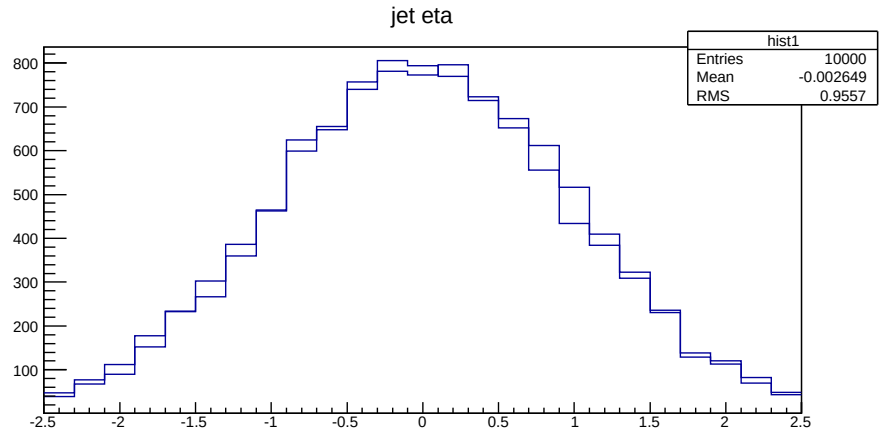
```
int TestCode(){

  // define canvas for output plot
  TCanvas  *c1      = new TCanvas("c1","The FillRandom example", 200, 10, 700, 700);

  // define 1D histogram with 25 bins and range -2.5 to 2.5
  TH1D     *fHisto1 = new TH1D("hist1", "jet eta", 25, -2.5, 2.5);
  TH1D     *fHisto2 = new TH1D("hist2", "jet eta", 25, -2.5, 2.5);

  // initialize random generator
  TRandom3 *fRand   = new TRandom3();

  // seed seed to system time: 0
  fRand -> SetSeed(0);

  int stats1 = 10000;

  // fill histograms with 10000 random values
  for(int i = 0; i < stats1; ++i){

    double value1 = fRand -> Gaus(0.0, 1.0);
    double value2 = fRand -> Gaus(0.0, 1.0);

    fHisto1 -> Fill(value1);
    fHisto2 -> Fill(value2);

  }

  // draw both histogram to canvas
  fHisto1 -> Draw();
  fHisto2 -> Draw("SAME");  // print both distributions in one canvas

  // print canvas
  c1      -> Print();

}
```

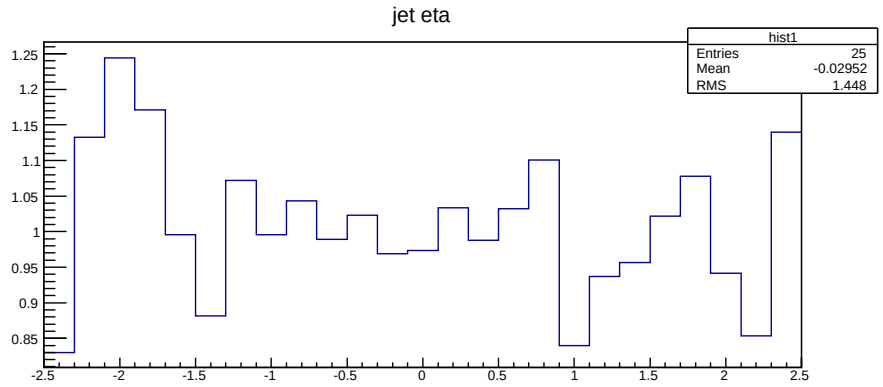


➔ have to specify Draw("SAME") when drawing the second histogram

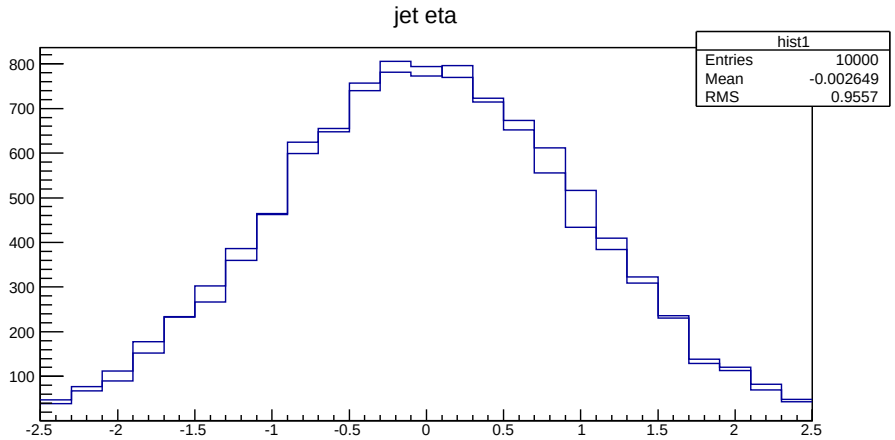
Now want to compare the two shapes: h1 → Divide(h2)

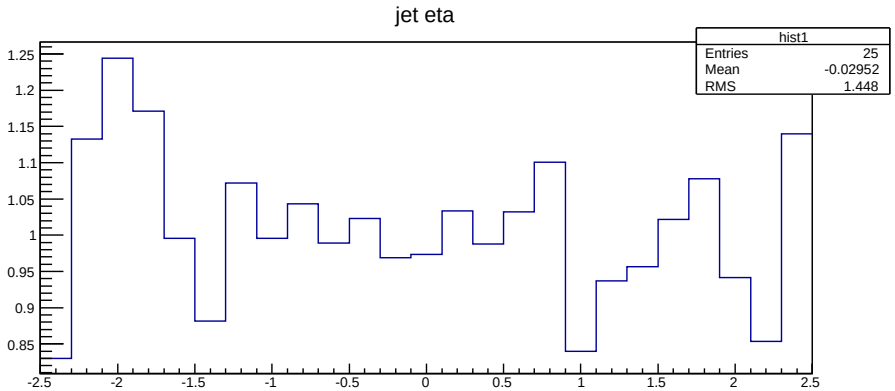
```
int TestCode(){

  // define canvas for output plot
  TCanvas  *c1      = new TCanvas("c1","The FillRandom example", 200, 10, 700, 700)

  pad1 = new TPad("pad1","main plot",  0.05,0.50,0.95,0.95,21);
  pad2 = new TPad("pad2","ratio plot", 0.05,0.05,0.95,0.45,21);
  pad1->SetFillColor(0);
  pad2->SetFillColor(0);
  pad1->Draw();
  pad2->Draw();
  pad1->cd(); // go not into first pad and draw the histograms

  // define 1D histogram with 25 bins and range -2.5 to 2.5
  TH1D     *fHisto1 = new TH1D("hist1", "jet eta", 25, -2.5, 2.5);
  TH1D     *fHisto2 = new TH1D("hist2", "jet eta", 25, -2.5, 2.5);

  // initialize random generator
  TRandom3 *fRand  = new TRandom3();

  fRand -> SetSeed(0);  // seed seed to system time: 0

  int stats1 = 10000;

  // fill histograms with 10000 random values
  for(int i = 0; i < stats1; ++i){

    double value1 = fRand -> Gaus(0.0, 1.0);
    double value2 = fRand -> Gaus(0.0, 1.0);

    fHisto1 -> Fill(value1);
    fHisto2 -> Fill(value2);

  }

  // draw both histogram to canvas
  fHisto1 -> Draw();
  fHisto2 -> Draw("SAME");  // print both distributions in one canvas

  pad2->cd(); // go now to second pad and get the ratio plot

  TH1D *fRatio = (TH1D*) fHisto1 -> Clone(); // clone now first histo
  fRatio -> Divide(fHisto2); // build the ratio of the two histograms!
  fRatio -> Draw();

  // print canvas
  c1     -> Print();
}
```





Look at our simple plot: good enough?





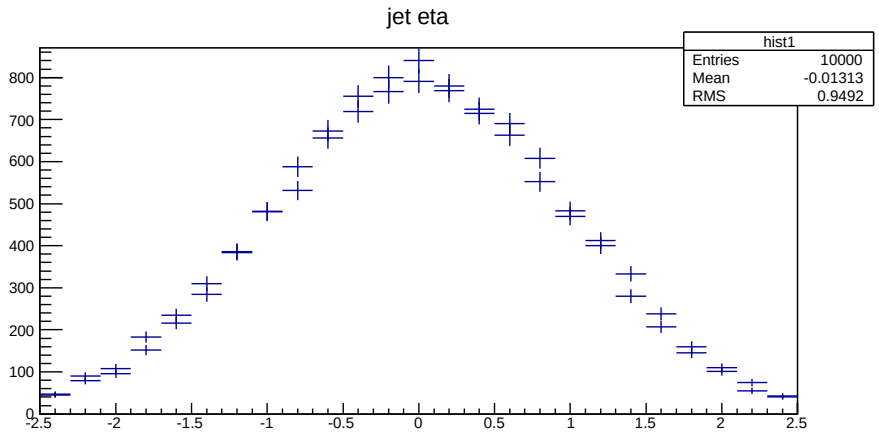
➔ No! See **at least 8 things** that are unclear/missing!

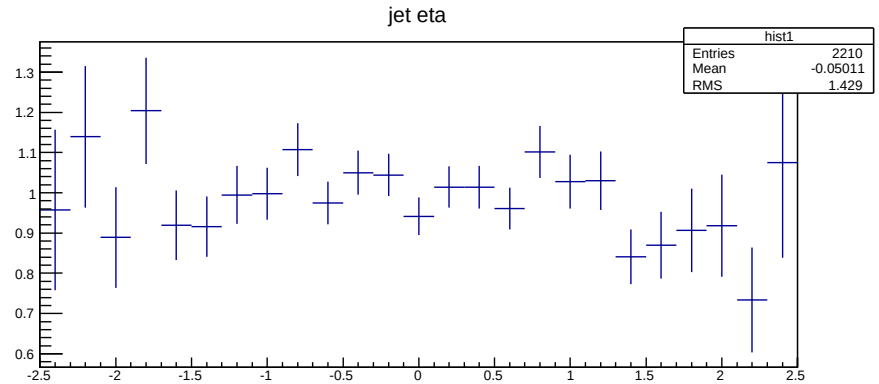
How to make now a good plot?

- has to be informative
- not too crowded, but complete!
- good, unambiguous colour code
- labels, titles, legend: good text size!
- good axis labels and titles: do not forget units!
- which analysis channel? how many jets/btags?
- which experiment, which luminosity?

Now want to show statistical uncertainty

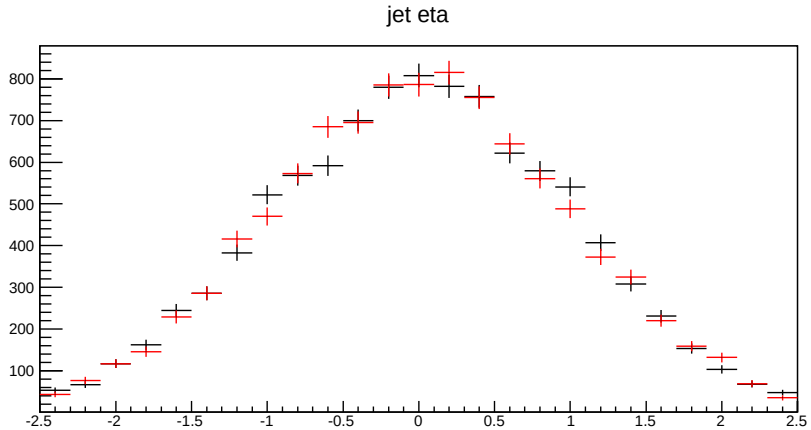
```
int TestCode(){

 // define canvas for output plot
 TCanvas  *c1      = new TCanvas("c1","The FillRandom example", 200, 10, 700, 700);

 pad1 = new TPad("pad1","main plot",  0.05,0.50,0.95,0.95,21);
 pad2 = new TPad("pad2","ratio plot", 0.05,0.05,0.95,0.45,21);
 pad1->SetFillColor(0);
 pad2->SetFillColor(0);
 pad1->Draw();
 pad2->Draw();
 pad1->cd(); // go not into first pad and draw the histograms

 // define 1D histogram with 25 bins and range -2.5 to 2.5
 TH1D     *fHisto1 = new TH1D("hist1", "jet eta", 25, -2.5, 2.5);
 TH1D     *fHisto2 = new TH1D("hist2", "jet eta", 25, -2.5, 2.5);

 // we want stat. uncertainties to be shown!
 fHisto1 -> Sumw2();
 fHisto2 -> Sumw2();
```

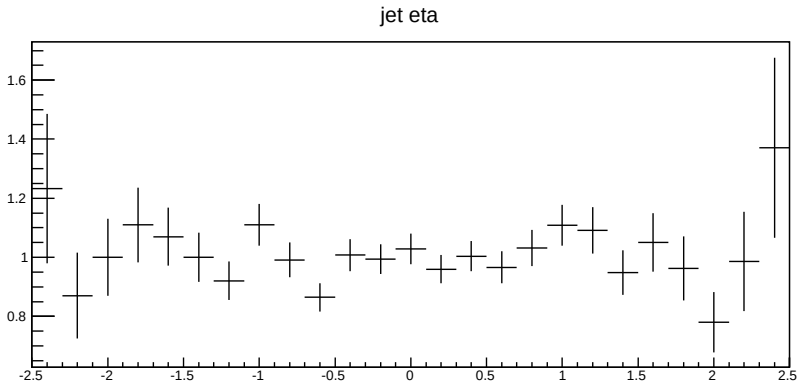




Now want to remove statistics box

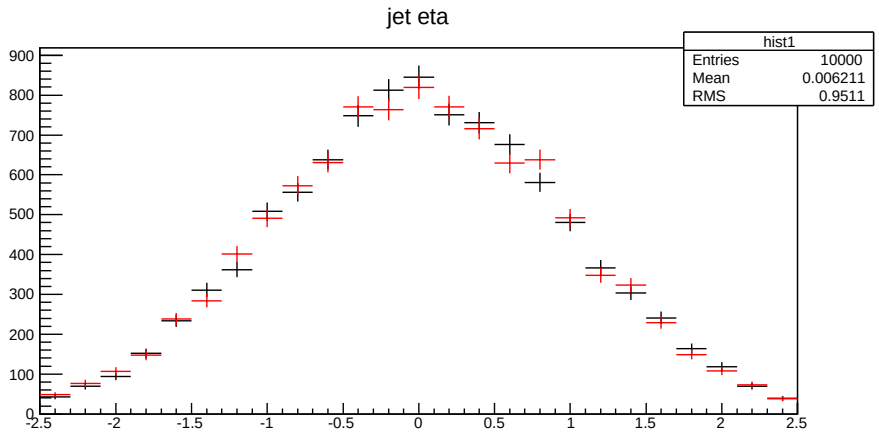
```
gStyle->SetOptStat(0); // remove the stats box

TH1D *fRatio = (TH1D*) fHisto1 -> Clone(); // clone now first histo
fRatio -> Divide(fHisto2); // build the ratio of the two histograms!
fRatio -> Draw();

// print canvas
c1     -> Print();
```

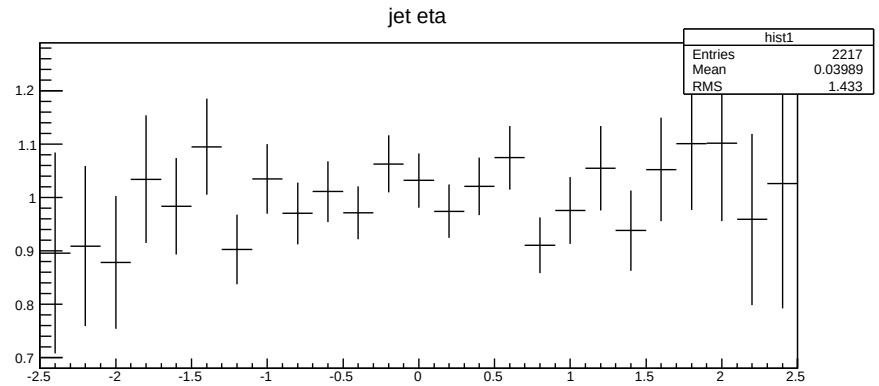




Now want proper colour code

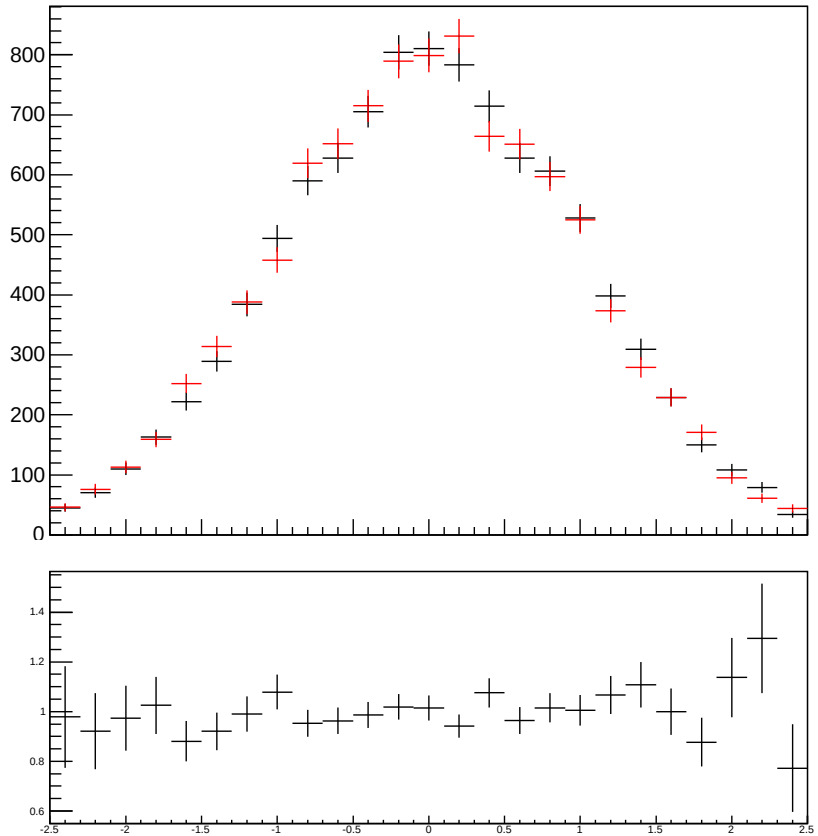
```
// use different line colors
fHisto1 -> SetLineColor(kBlack);
fHisto2 -> SetLineColor(kRed);

// draw both histogram to canvas
fHisto1 -> Draw();
fHisto2 -> Draw("SAME");  // print both distributions in
```





Now want better ratio plot

```
// define canvas for output plot
TCanvas  *c1      = new TCanvas("c1","The FillRandom example", 200, 10, 700, 700);

pad1 = new TPad("pad1","main plot",  0.05,0.30,0.95,0.95,21);  // make upper hist bigger
pad2 = new TPad("pad2","ratio plot", 0.05,0.05,0.95,0.36,21);  // make lower hist smaller
pad1->SetFillColor(0);
pad2->SetFillColor(0);
pad1->Draw();
pad2->Draw();
pad1->cd(); // go not into first pad and draw the histograms
```

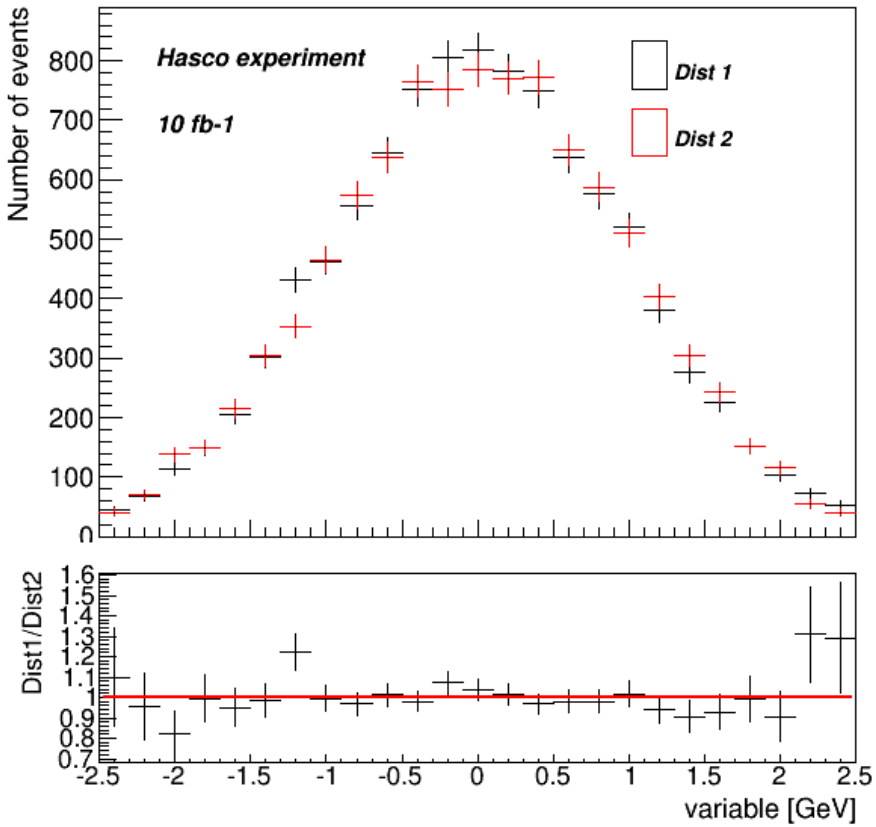


Add more information to plot

fixed now:

- add axis title
- adjust label and title sizes
- add line at y = 1 to ratio plot
- add Legend
- add label for experiment and sample stats

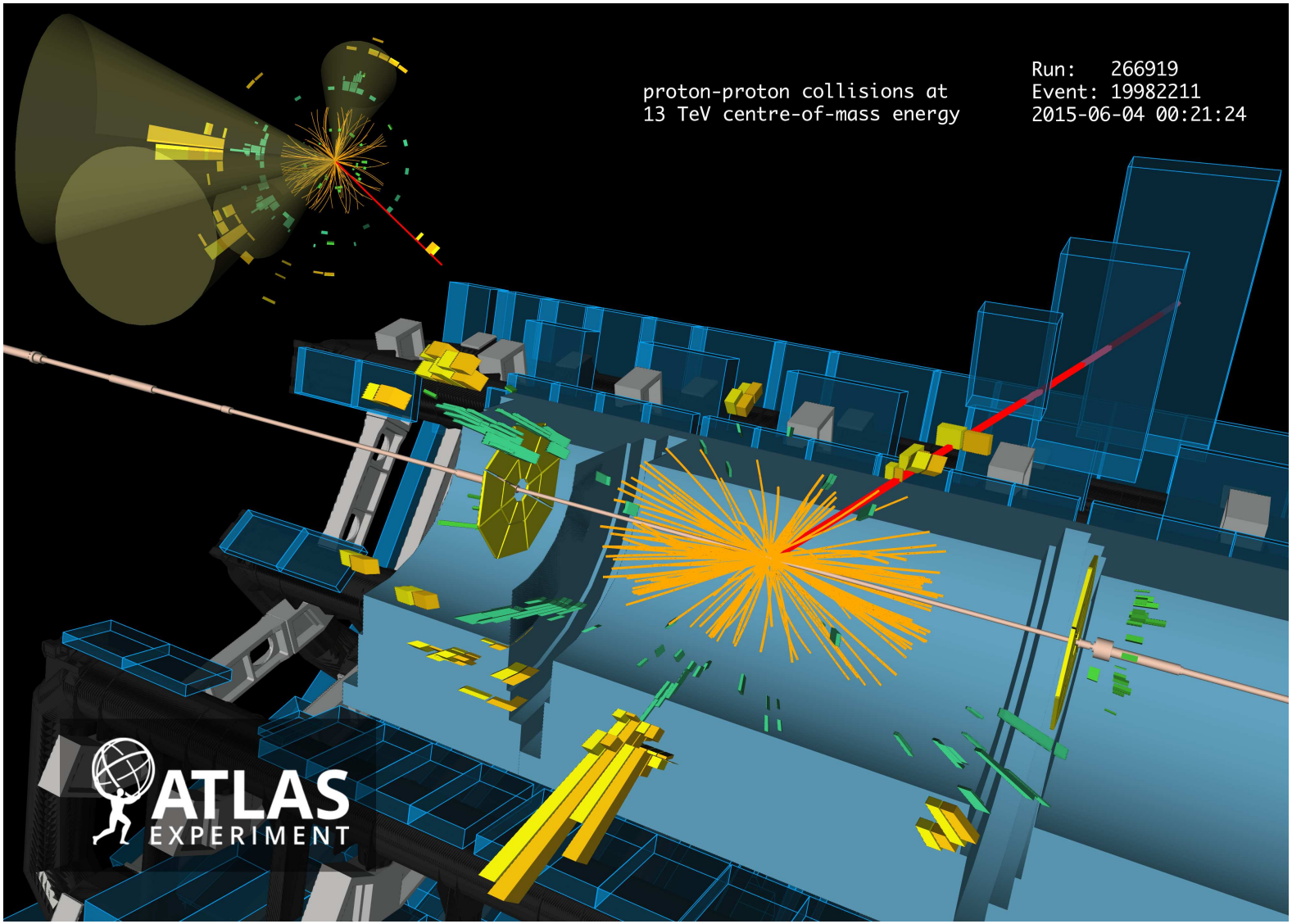
➔ all of this will be in tomorrows tutorial



What is a flat ntuple?

- store all info we have in ROOT **Tree**
- have a **branch** for each variable
- a branch can contain an integer, float, double, vector of floats, vector of vector...
- "flat" ntuple: easily browsable
- simplest object if you want to do your final analysis
- contains object kinematics (four vectors, scale factors, event weights...)

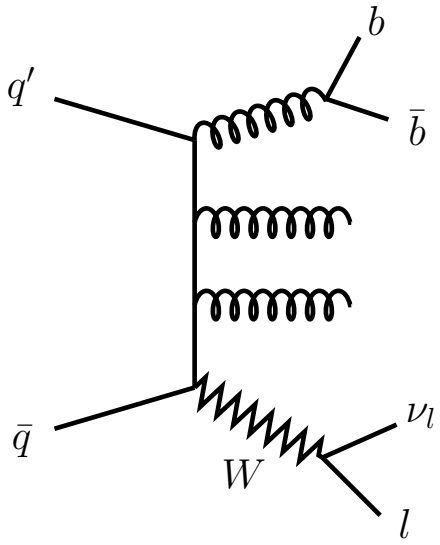
A pretty HEP event



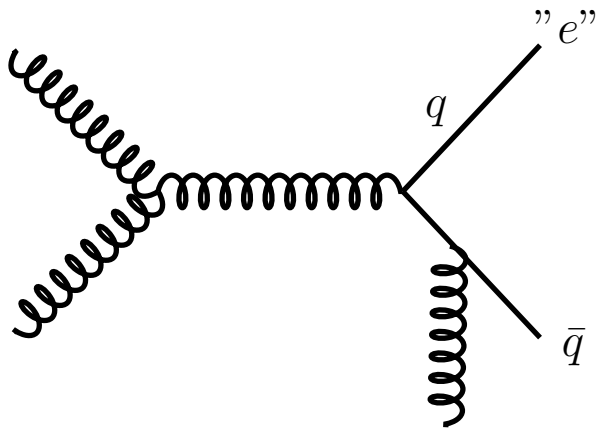
... but it does not always look that clean!

Dominant Background Processes (l+jets)

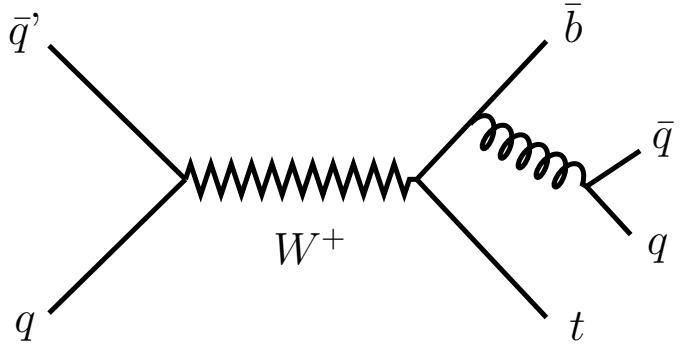
W+jets:



Misidentified leptons:



Single top:



Smaller contributions:

- Z+jets
- Diboson

➔ Need a lot of variables in a handy format

Event selection: lepton+jets

Missing E_T:

Miss. $E_T > 20$ GeV

Electrons:

- $E_T > 25$ GeV
- $|\eta| < 2.47$
- isolated
- exclude crack region

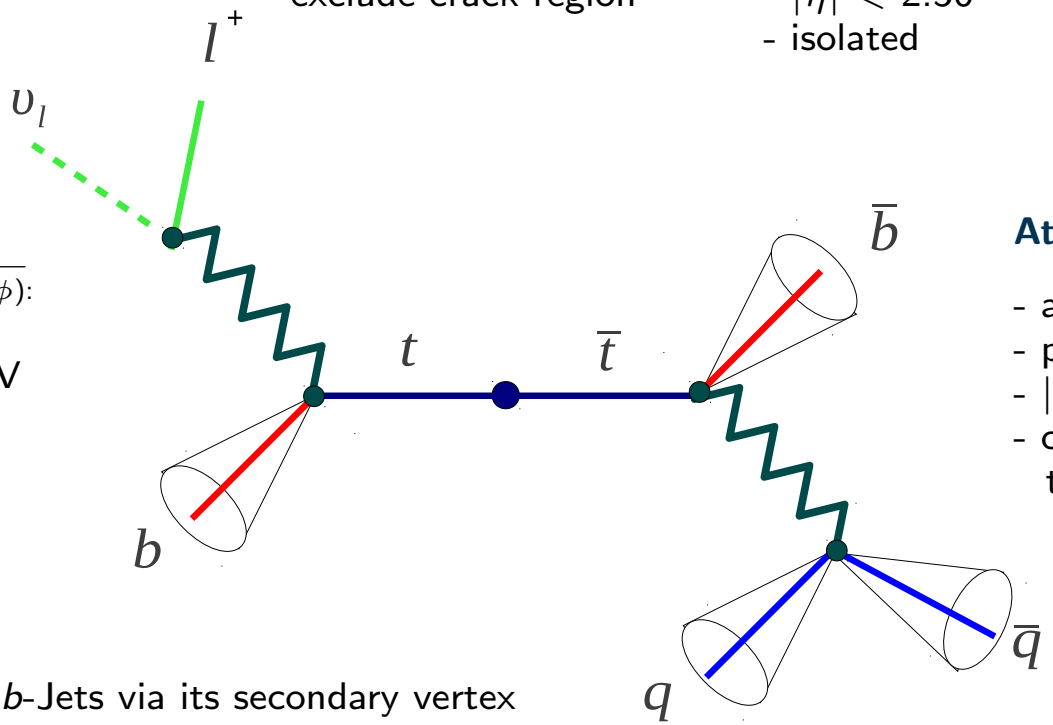
Muons:

- combined Muons (tracker+spectrometer)
- $p_T > 20$ GeV
- $|\eta| < 2.50$
- isolated

transverse W mass:

$m_{T,W} = \sqrt{2p_T^l p_T^\nu (1 - \cos(\Delta\phi))}$:

$m_{T,W} + E_T^{\text{miss}} > 60$ GeV



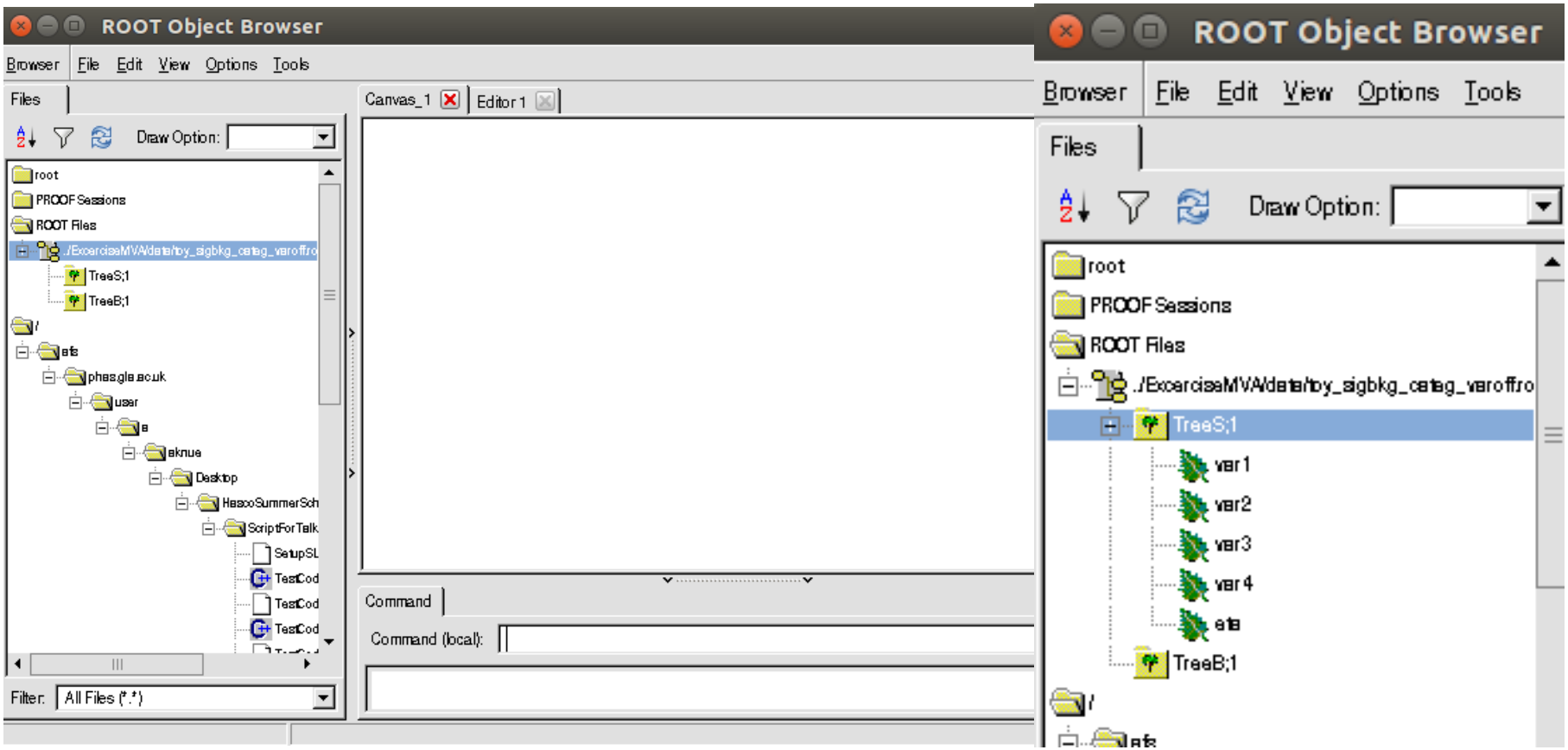
At least 4 Jets:

- anti-k_t jets ($\Delta R = 0.4$)
- $p_T > 25$ GeV
- $|\eta| < 2.5$
- cut on Jet Vertex Tagger to reduce pile-up influence

***b*-Jets:**

➔ identify *b*-Jets via its secondary vertex
➔ require at least one tagged jet
➔ use MV2 @ 77 % WP

Open tree in browser



➔ do: root -l FileName.root

➔ and then: new TBrowser

I was given a large ROOT Tree to analyse, what now?

Make structure to read it

➔ magical command: **MakeClass**

Example:

- filename: example.root, tree name: TreeS
- type: root example.root
- then type: TreeS→MakeClass("TestTree")
- you get: TestTree.h and TestTree.C

```
root [1] TBrowser b
root [2] TreeS->MakeClass("TestTree")
Info in <TTreePlayer::MakeClass>: Files: TestTree.h and TestTree.C generated from TTree: TreeS
(Int_t)0
root [3]
```

Output header file: TestTree.h

```
File Edit Options Buffers Tools C Help
//////////////////////////////////////////////////////////
// This class has been automatically generated on
// Wed Jul 22 17:37:08 2015 by ROOT version 5.34/10
// from TTree TreeS/TreeS
// found on file: ../ExcerciseMVA/data/toy_sigbkg_categ_varoff.root
//////////////////////////////////////////////////////////

#ifndef TestTree_h
#define TestTree_h

#include <TROOT.h>
#include <TChain.h>
#include <TFile.h>

// Header file for the classes stored in the TTree if any.

// Fixed size dimensions of array or collections stored in the TTree if any.

class TestTree {
public :
   TTree          *fChain;   //!pointer to the analyzed TTree or TChain
   Int_t           fCurrent; //!current Tree number in a TChain

   // Declaration of leaf types
   Float_t         var1;
   Float_t         var2;
   Float_t         var3;
   Float_t         var4;
   Float_t         eta;

   // List of branches
   TBranch        *b_var1;   //!
   TBranch        *b_var2;   //!
   TBranch        *b_var3;   //!
   TBranch        *b_var4;   //!
   TBranch        *b_eta;   //!

   TestTree(TTree *tree=0);
   virtual ~TestTree();
   virtual Int_t    Cut(Long64_t entry);
   virtual Int_t    GetEntry(Long64_t entry);
   virtual Long64_t LoadTree(Long64_t entry);
   virtual void     Init(TTree *tree);
   virtual void     Loop();
   virtual Bool_t   Notify();
   virtual void     Show(Long64_t entry = -1);
};
```

Output macro: TestTree.C

```
File Edit Options Buffers Tools C++ Help
#define TestTree_cxx
#include "TestTree.h"
#include <TH2.h>
#include <TStyle.h>
#include <TCanvas.h>

void TestTree::Loop()
{
//   In a ROOT session, you can do:
//      Root > .L TestTree.C
//      Root > TestTree t
//      Root > t.GetEntry(12); // Fill t data members with entry number 12
//      Root > t.Show();       // Show values of entry 12
//      Root > t.Show(16);     // Read and show values of entry 16
//      Root > t.Loop();       // Loop on all entries
//

//     This is the loop skeleton where:
//    jentry is the global entry number in the chain
//    ientry is the entry number in the current Tree
//  Note that the argument to GetEntry must be:
//    jentry for TChain::GetEntry
//    ientry for TTree::GetEntry and TBranch::GetEntry
//
//       To read only selected branches, Insert statements like:
// METHOD1:
//    fChain->SetBranchStatus("*",0);  // disable all branches
//    fChain->SetBranchStatus("branchname",1);  // activate branchname
// METHOD2: replace line
//    fChain->GetEntry(jentry);       //read all branches
//by  b_branchname->GetEntry(ientry); //read only this branch
   if (fChain == 0) return;

   Long64_t nentries = fChain->GetEntriesFast();

   Long64_t nbytes = 0, nb = 0;
   for (Long64_t jentry=0; jentry<nentries;jentry++) {
      Long64_t ientry = LoadTree(jentry);
      if (ientry < 0) break;
      nb = fChain->GetEntry(jentry);   nbytes += nb;
      // if (Cut(ientry) < 0) continue;
   }
}
```

Root Macro vs compiled code

up to now:

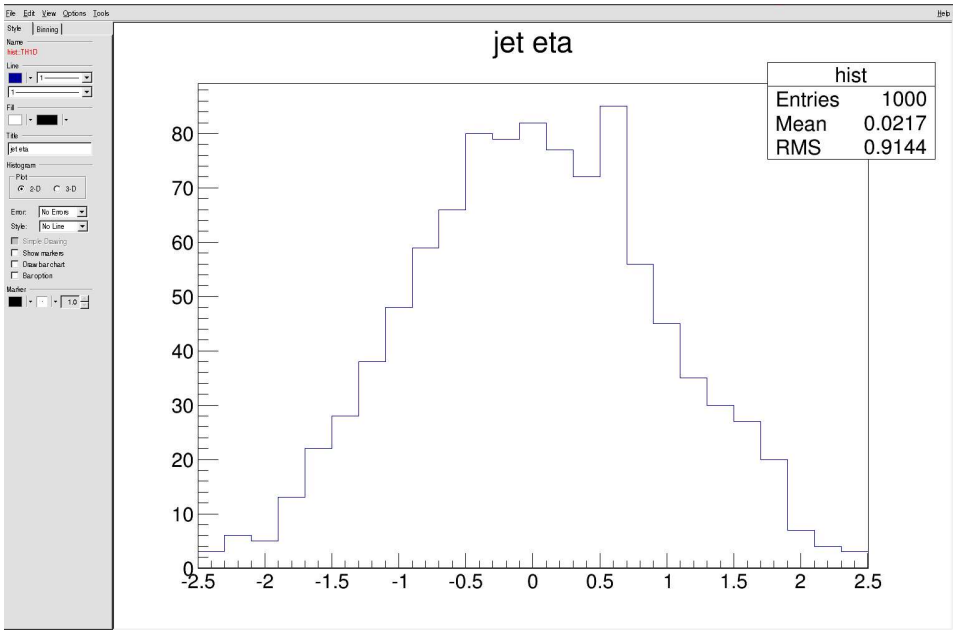
- only showed how to run a root macro with root MacroName.C
- fine for small tests, very slow for larger statistics

How to compile a root macro (using CINT, ACLIC)

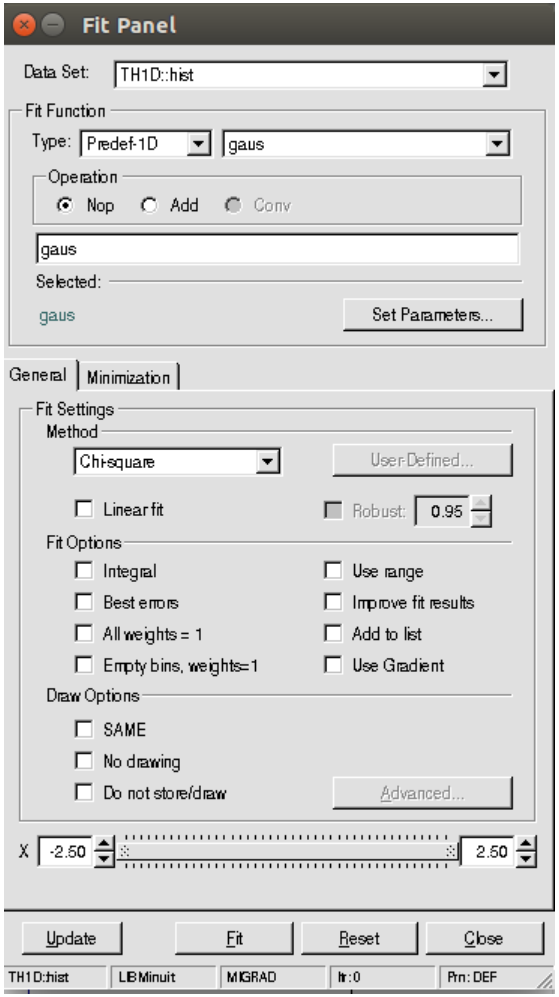
- open root in command line, then do: .L TestScript.C++
- have to make sure all neccessary non-root libraries (like iostream, string etc) are included
- **but: better include the libraries in a proper C++ class and use a Makefile!**

The Root browser/editor

- want to work with a histogram that has been stored in a .root file
- open it with root: root HistoFile.root
- then click on histogram in the file



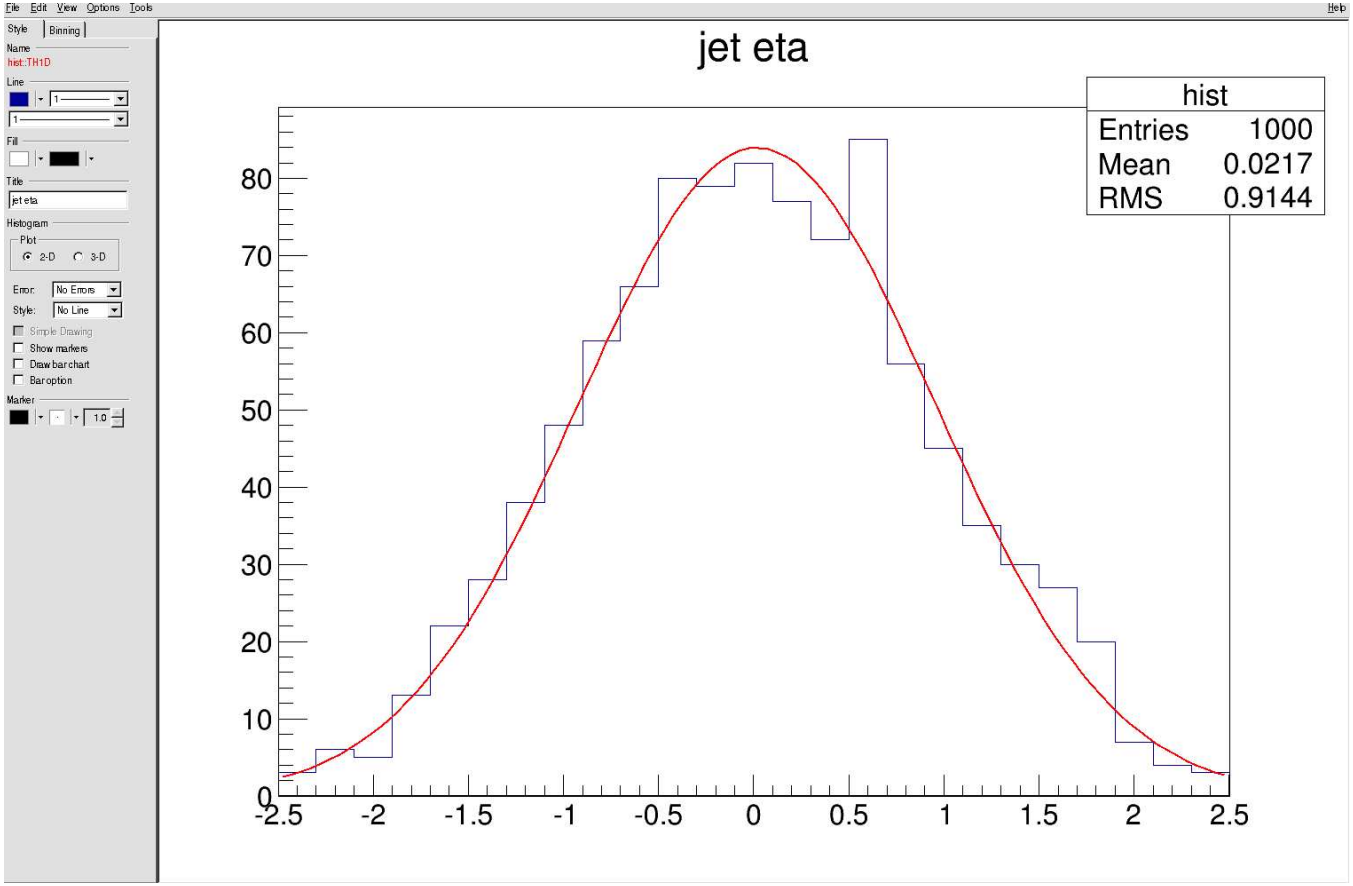
The Root browser/editor




```
Info in <TCanvas::SaveAs>: ROOT file TestOutput.root has been created
root [1]  FCN=12.672 FROM MIGRAD    STATUS=CONVERGED      57 CALLS          58 TO
                     EDM=3.6588e-11    STRATEGY= 1      ERROR MATRIX ACCURATE
  EXT PARAMETER                                          STEP         FIRST
  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
   1  Constant     8.39369e+01   3.32320e+00   4.84754e-03   2.82605e-06
   2  Mean         1.71986e-02   3.08937e-02   5.57613e-05   5.67416e-05
   3  Sigma        9.36009e-01   2.32333e-02   1.22025e-05   1.07896e-03
```

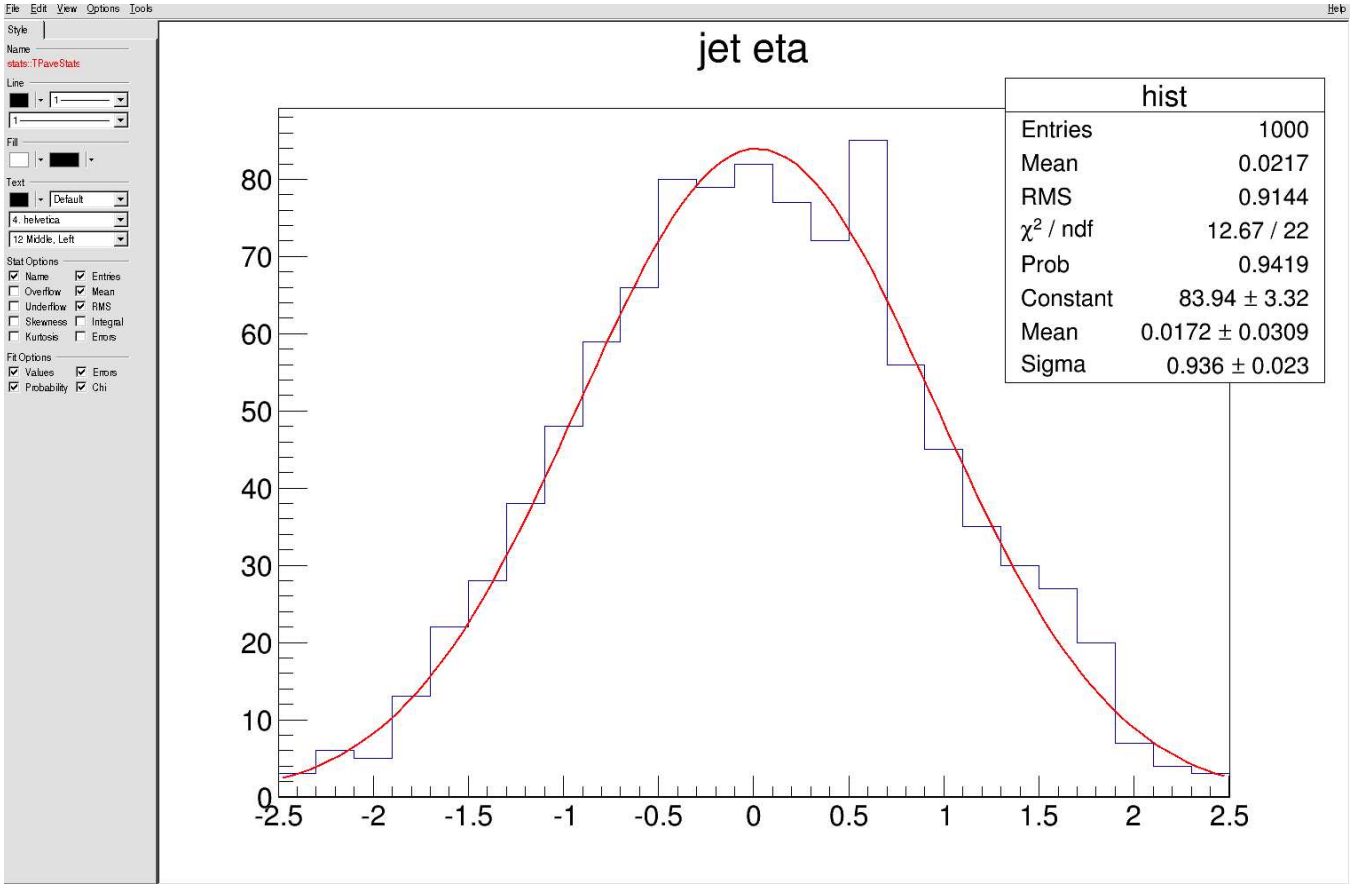
- open histogram in TBrowser
- click on: View → Editor
- then: Tools → FitPanel
- choose fit function
- press "Fit" button
- get fit parameters in terminal

The Root browser/editor



➔ Now one gets the function from the fit result on top of the histogram.

The Root browser/editor

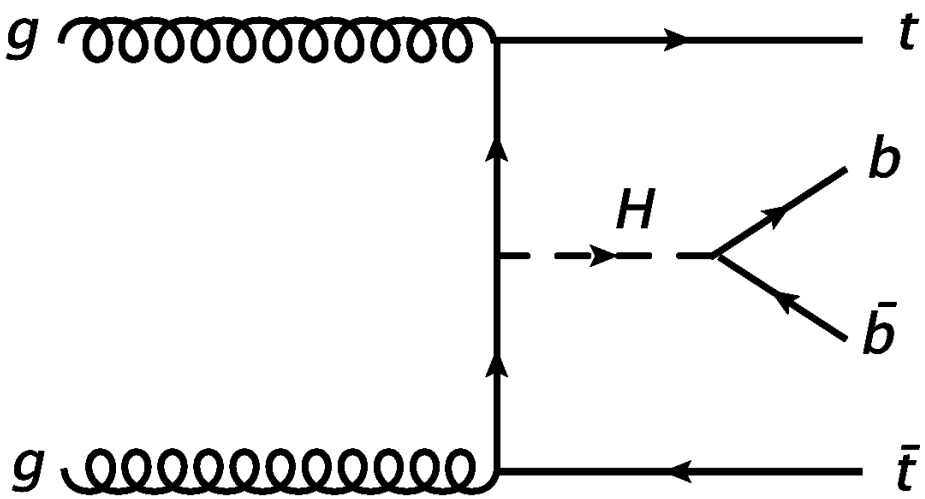


➔ can choose in panel on left side which results should be shown in stats box.

Run Multivariate Analysis

- available in ROOT in TMVA package
- what is a multivariate analysis?
- want to distinguish signal from background, one variable not powerful enough
- not too many, but enough variables (balance between separation power and running time)
- check for overtraining
- start with example!

Example: ttH analysis with $H \rightarrow b\bar{b}$ ▸ Paper

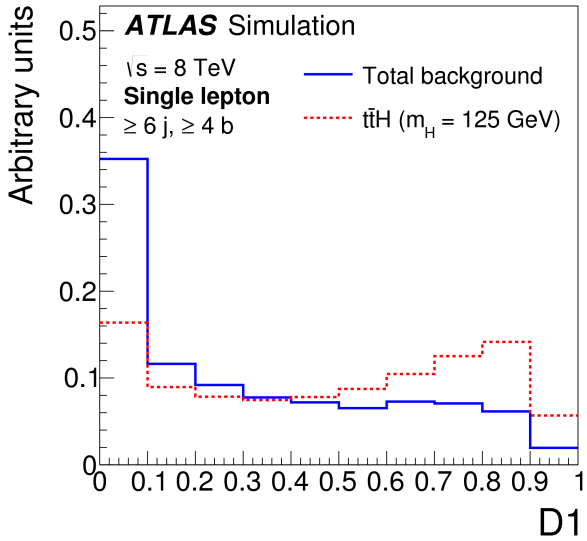


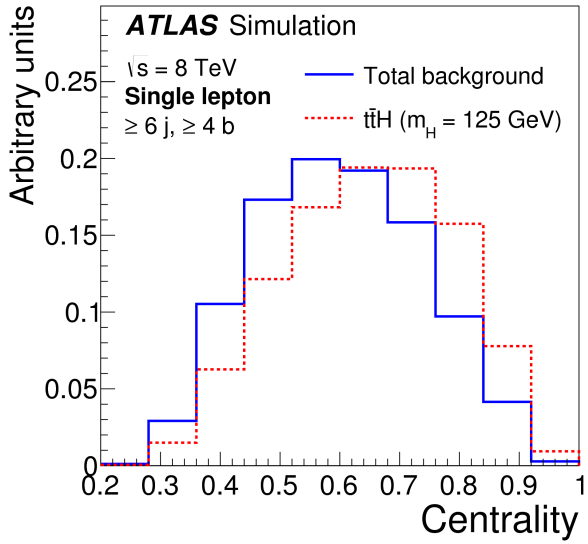
➔ but: if a gluon is radiated instead of a Higgs, the final state is identical!

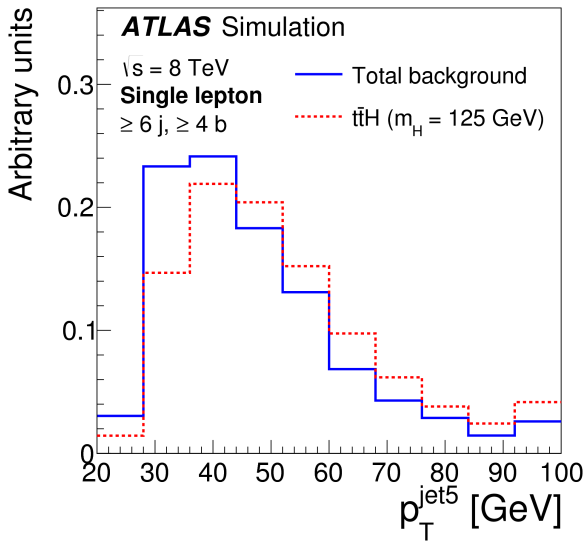
➔ large irreducible background, need to find variables to distinguish the two processes

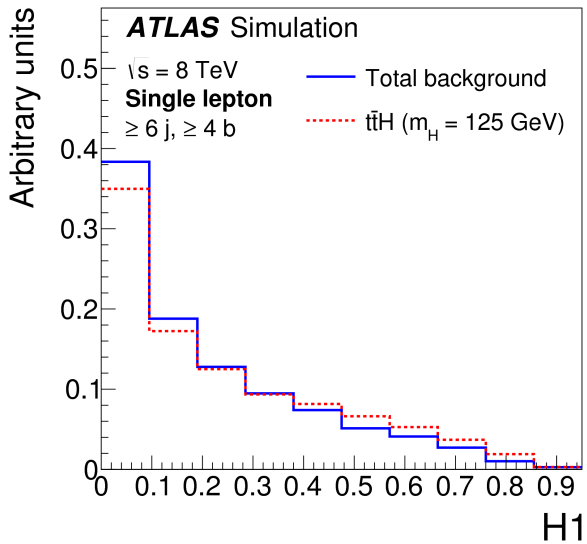
Signal region: 6 jets with 4 *b*-tags

Paper



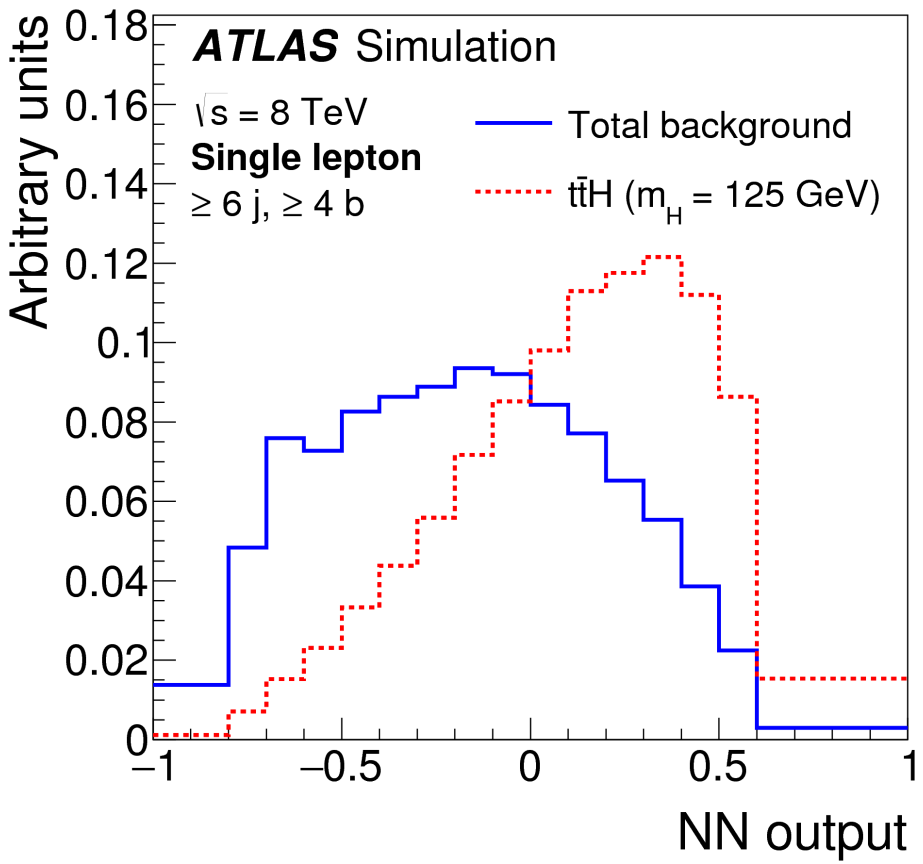






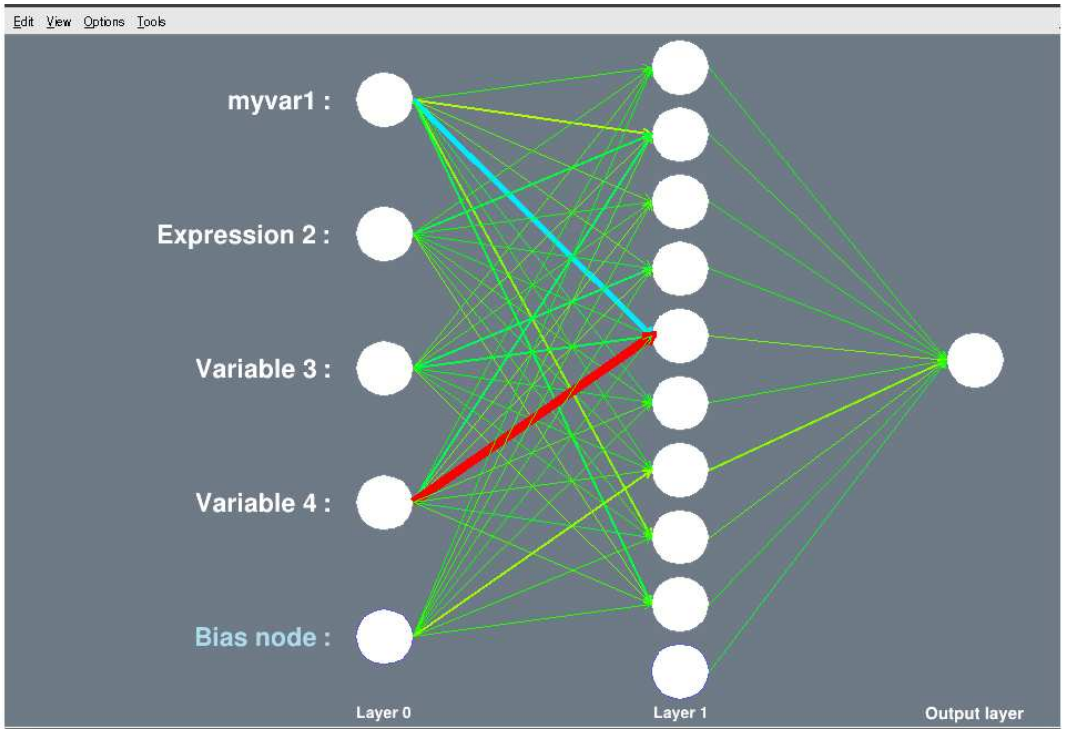
➔ some separation in each distribution: take all this info and make new variable!

Final analysis discriminant



➔ use information of 10 different variables: much better separation

Neural networks

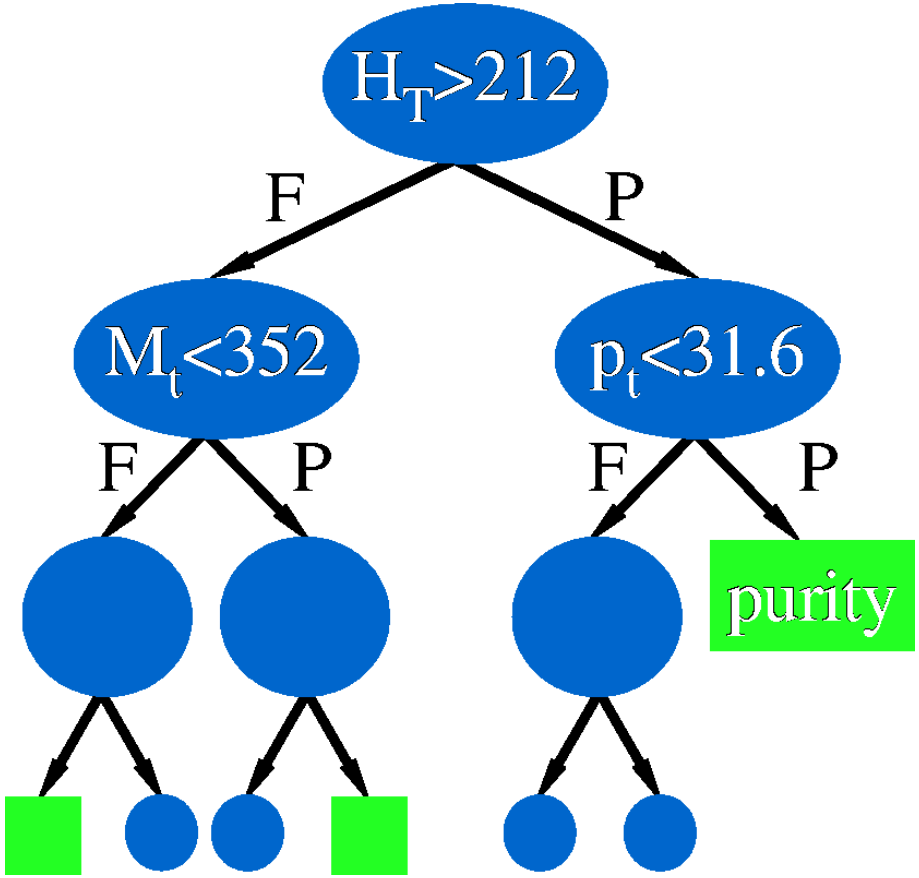


➔ input layer: as many nodes as input variables

➔ next layer: combine variables, NN can "learn"

Example: Boosted decision trees

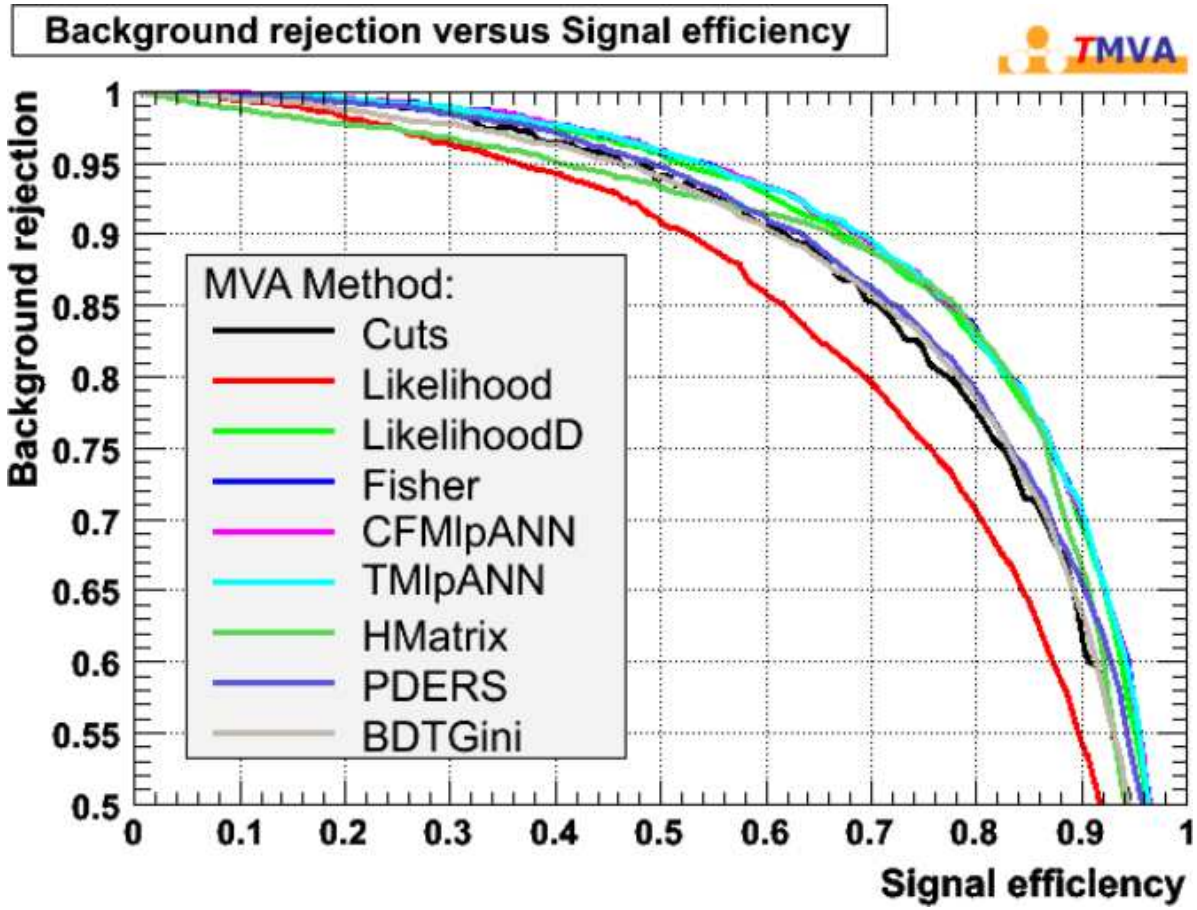
▸ Plot from here



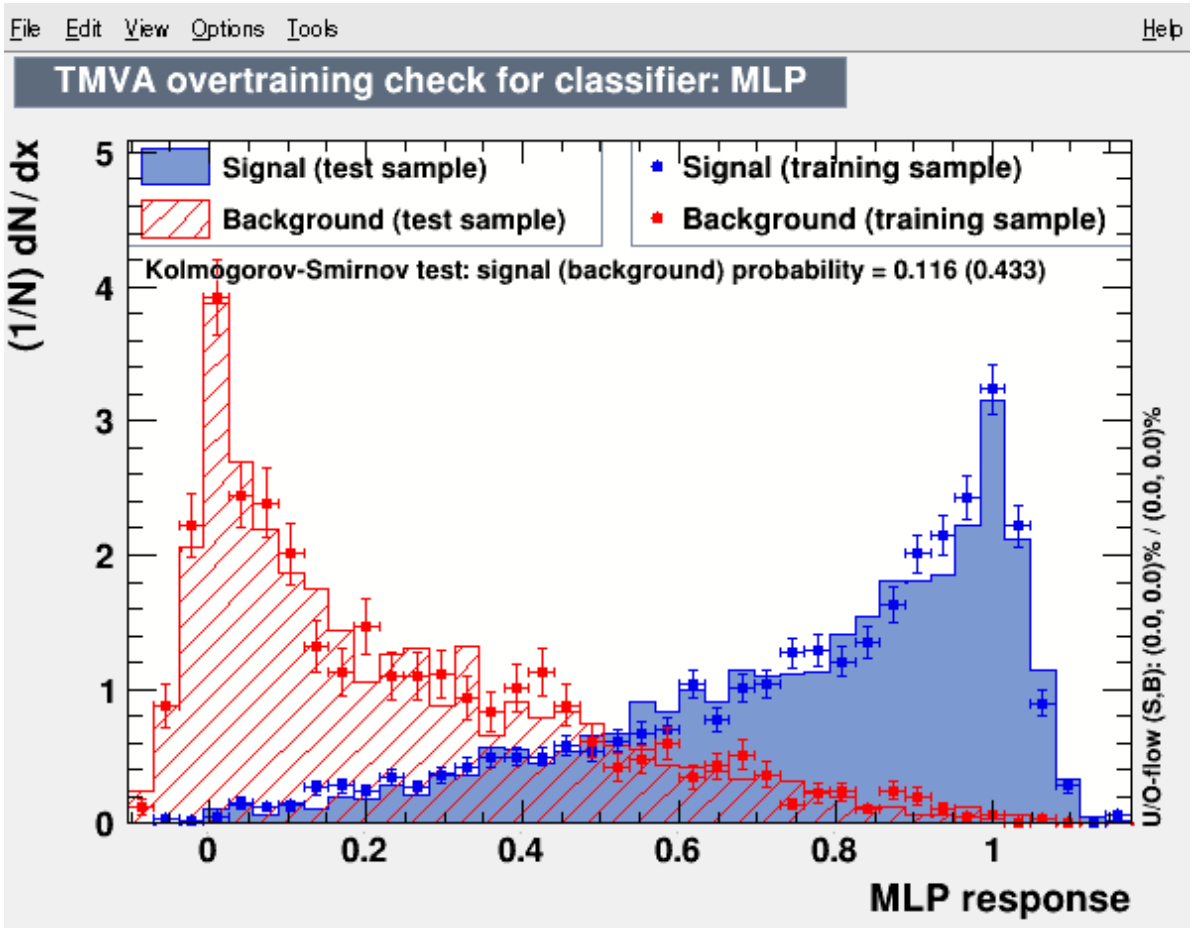
➔ but many more methods can be tested, lots implemented in TMVA

Compare different methods directly

▸ Plot from here



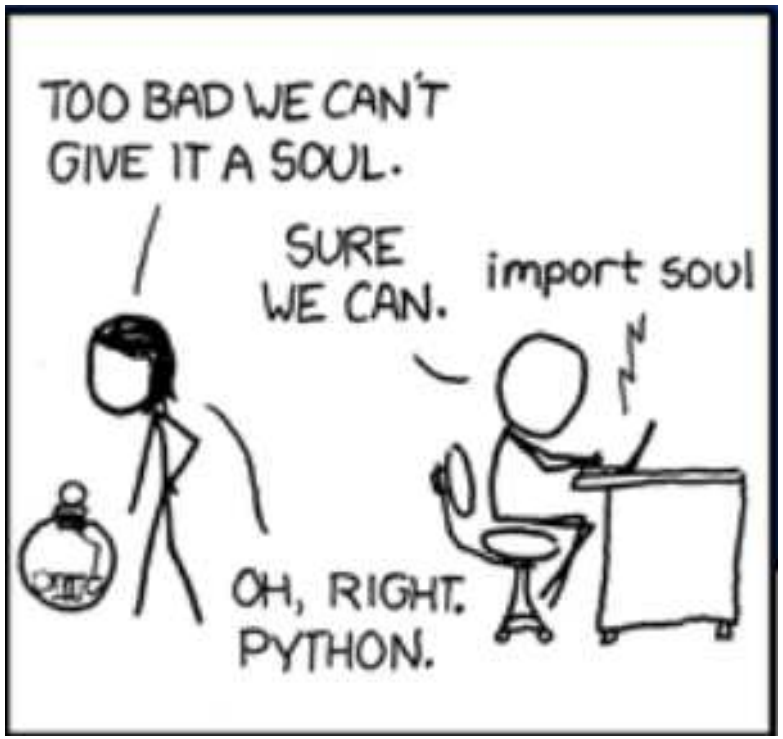
Check for overtraining!



➔ overtraining: NN takes statistical fluctuations into account

➔ use only half of sample for training, then use other half to test

Pyroot: use ROOT in python script



➔ this will be covered in one of the excercises tomorrow

➔ more documentation here

Summary

- need to store a lot of information
- info has to be easily accessible/modifyable
- need to be able to make pretty plots
- this and much more provided by ROOT
- Warning: has some annoying functionalities that you might experience soon!
- (not fortran, sorry PAW fans!)

Ready for some actual work?

➔ See you for the tutorial tomorrow!

Exercises

1. Find Higgs boson in $gg \to H \to \gamma\gamma$ channel and make money plot.
2. Read trees and make correlation plots with pyROOT.
3. Train different MVA and compare the performance.