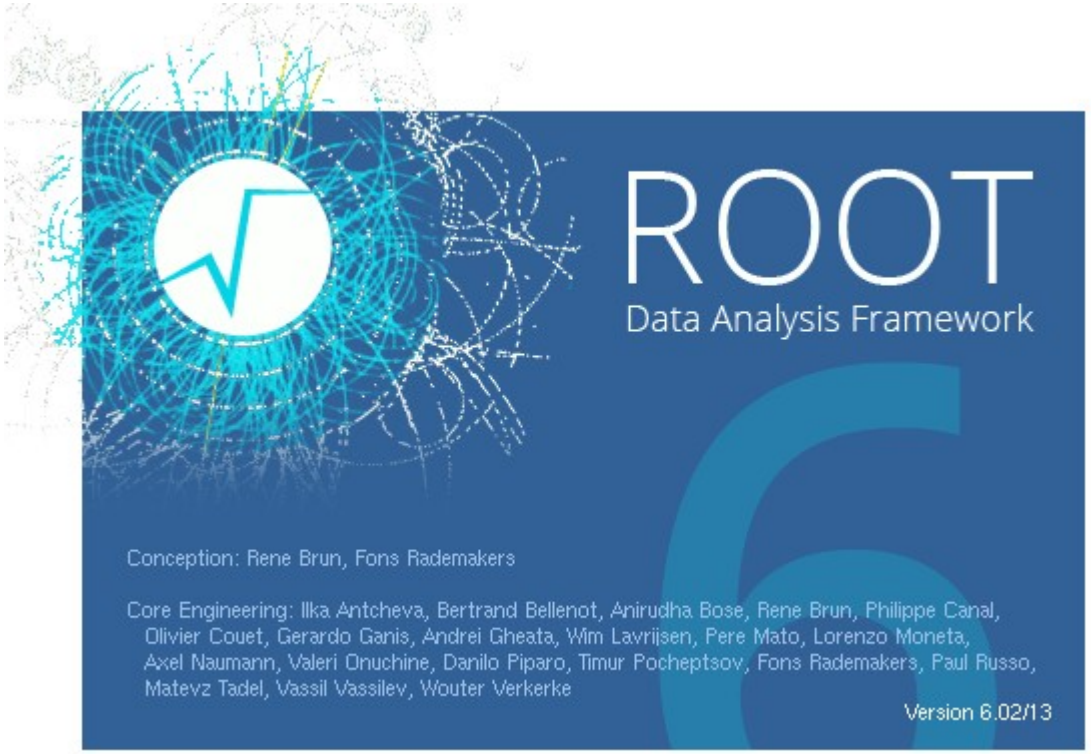
Introduction to ROOT and TMVA

Fredrik Oljemark, PhD



Contents:

- In these slides I will present a general overview and few key practical examples on the ROOT toolkit and one plugin to it, TMVA.

- We cannot cover exhaustively all the aspects but at the end you will know what these programs are used for and you will have a feeling on how you can use them.

- First Part:
  - Introduction to ROOT: how to run it, how to use the files to store data and read it, how to perform fits.

- Second Part:
  - Short introduction to TMVA: structure of the software, and some example code, with explanations.

FIRST PART:

INTRODUCTION TO ROOT

What do we do with ROOT?

Root is a C++ based analysis framework widely used in HEP. It provides a powerful and easy to use environment to perform:

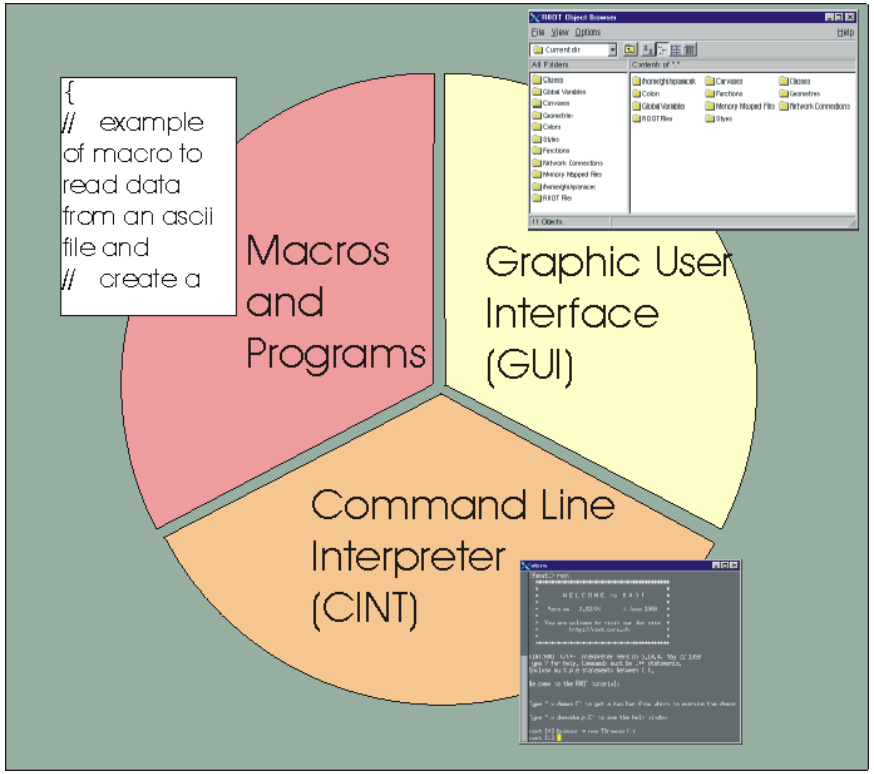
- Histograms and fits
- Graph, scatter plots (2D, 3D), GUI applications
- **I/O** files
  - Production of Ntuple (TTrees), see later.

Root helps physicist to perform data analysis thanks to:

- A user interface
  - GUI: Browsers, Panels, Tree Viewer
  - Command line interface: C++ interpreter (CINT)
  - Python interface to ROOT, see https://root.cern/manual/python/
  - Script processing (C++ compiled)
  - Very many statistical and mathematical libraries

Interacting with ROOT

Compiled C++ programs are much faster than interpreted ones. Libraries and executables can be created and used externally to the root framework



- **Interactive GUI (buttons, graphical menu…)**
  - Open/Browse root file
  - Display histograms content rebin/fits Change scales
  - Online analysis of the data (simple variable selection and content visualization

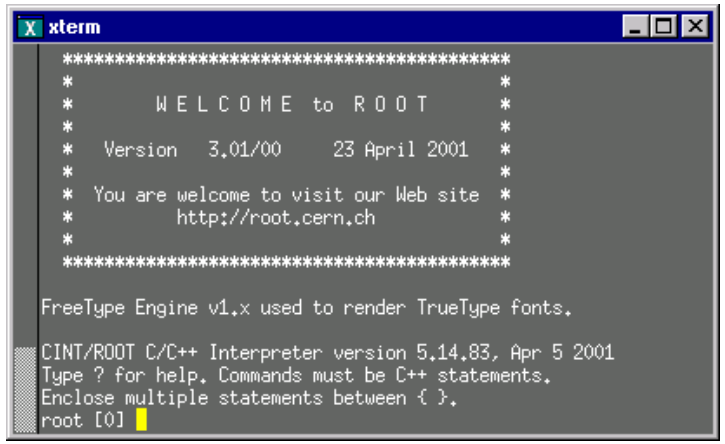
Root interactive command line CINT (C++ interpreter, idiosyncratic/more forgiving than compiler): mainly used for simple operations: display operations, basic fits of the histograms, histogram content check.

Example: start a new GUI session

1) Download and install root: https://root.cern/install/

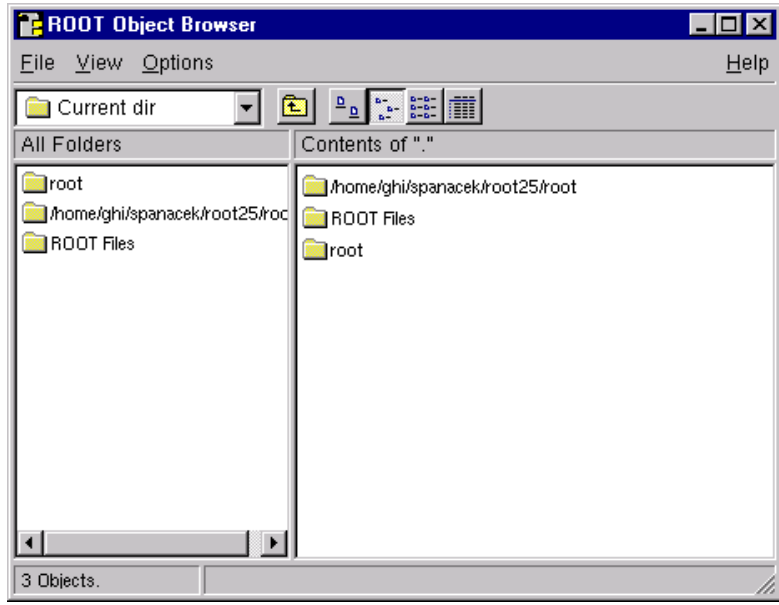
Call root to access the command line:

> `root`



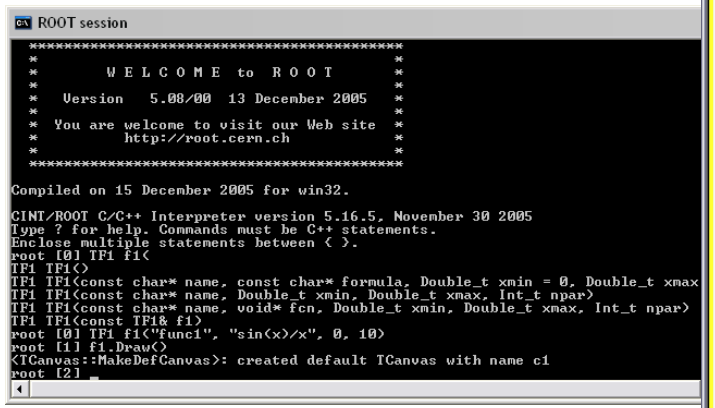
Open a new Browser:

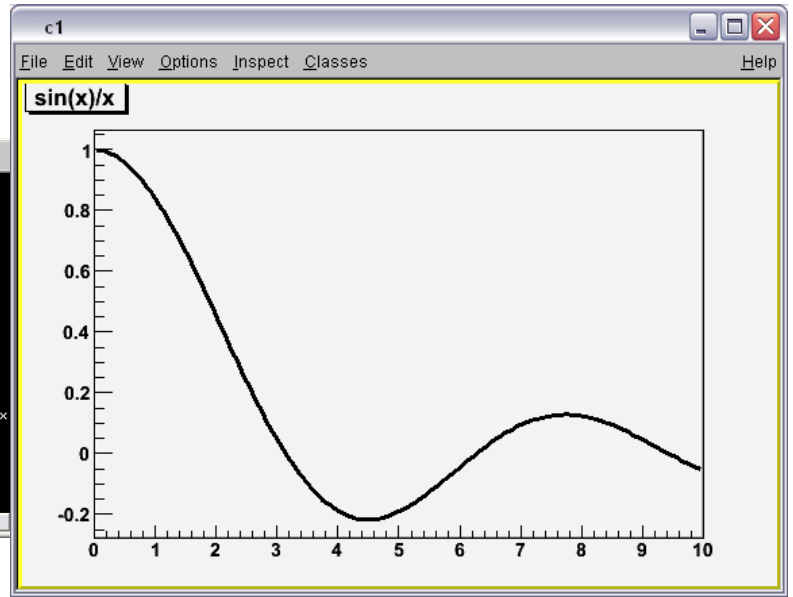
```
[ ] TBrowser b;
```



name formula range

```
[ ] TF1 f1("func1","sin(x)/x",0,10)
[ ] f1.Draw()
```

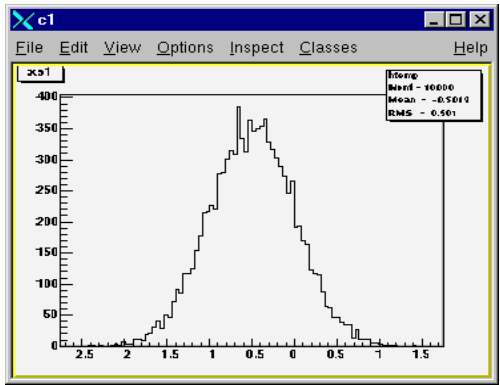




Draw a new function

or check an histogram already saved in a root data file: (double click in Tbrowser or):

```
[ ] Tfile* f= new TFile("namef.root")
[ ] //TFile f("namef.root","READ")
[ ] TH1F* histo=(TH1F*) f->Get("nameh")
[ ] histo->Draw()
```



Example: create a histogram according to a given underlying distribution and then fit it:

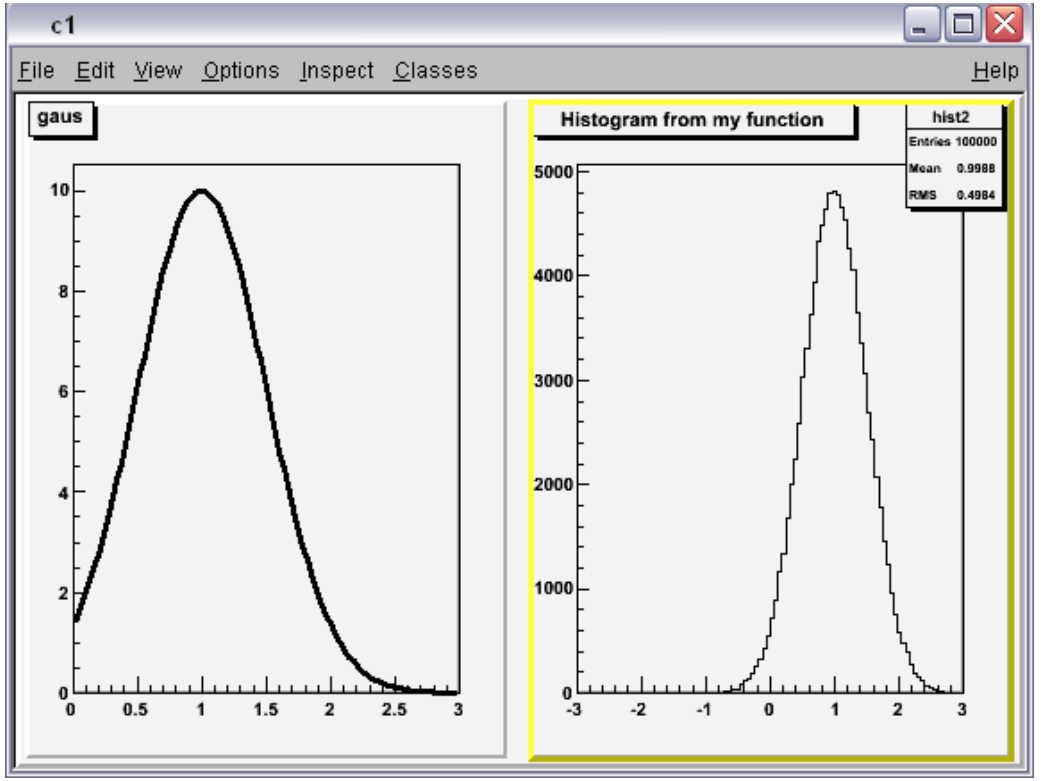
1- Generation

Gauss function:

$$par(0)e^{-0.5\left(\frac{x-par(1)}{par(2)}\right)^2}$$

```
[ ] TF1 myfunc("myfunc","gaus",0,3);
[ ] myfunc.SetParameters(10.,1.,0.5);
[ ] TCanvas c;
[ ] c.Divide(2,1);
[ ] c.cd(1);
[ ] myfunc.Draw();
[ ] TH1F h2("hist","Histo from my function",100,-3,3);
[ ] h2.FillRandom("myfunc",100000);
[ ] c.cd(2);
[ ] h2.Draw();
```

par(0)=10, par(1)= 1, par(2)= 0.5

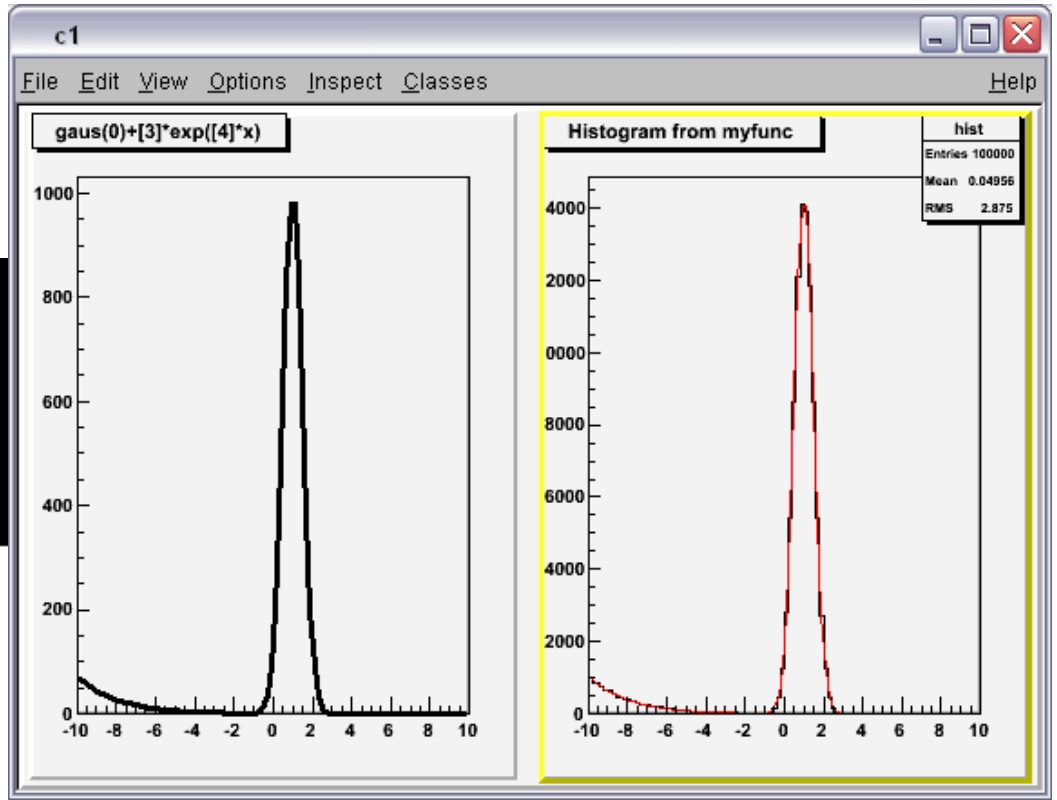


2- Fit

Fit a histogram with the function: $par(0)e^{-\left(\frac{x-par(1)}{par(2)}\right)^2} + par(3)e^{par(4)x}$

```
[ ] TF1 f1("myfunc","gaus(0)+[3]*exp([4]*x)",-10.,10.);
[ ] f1.SetParameters(1000.,1.,0.5,0.5,-0.5);
[ ] TH1F h1("hist","Histogram from myfunc",100,-10,10);
[ ] h1.FillRandom("myfunc",100000);
[ ] h1.Fit("myfunc");
[ ] //gStyle->SetOptFit(1) : fit parms in statbox
```

```
root [10] h1.Fit("myfunc")
 FCN=58.0027 FROM MIGRAD    STATUS=CONVERGED     538 CALLS         539 TOTAL
                     EDM=1.40487e-008    STRATEGY= 1  ERROR MATRIX UNCERTAINTY   2.1 per cent
  EXT PARAMETER                                   STEP         FIRST
  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
   1  p0           1.43217e+004   5.83272e+001
   2  p1           9.98341e-001   1.66154e-003
   3  p2           4.98132e-001   1.17738e-003
   4  p3           7.32169e+000   3.23833e-001
   5  p4          -4.96196e-001   5.34834e-003
(Int_t)0
root [11]
```



Compiling and executing macros

- Command line commands (as the example before) can be organized in single files.
- Depending on the complexity of the task, we can just need a set of command to be interpreted (macros) or we can create a shared library that can be executed or used by other programs (compiled)

  - Named scripts can be **interpreted** line by line (interpreter CINT)

```
root [3] .x myMacro.C;
```

  - **Or Compiled** to produce a shared library via ACLiC (then executable)

```
root [3] .L myMacro.C++; //always recompile
root [3] .L myMacro.C+;  //recompile only if necessary
root [3] .x myMacro.C++; //compile and execute
root [3] .L myMacro_C.so; //load the shared library
root [3] myMacro(); //execute the function of the library
root [3] .U myMacro_C.so; //unload the library
```

Example: execution of a macro "say.C"

```
void say(TString what="Hello")
{
  cout << what << endl;
```

```
root [3] .x say.C
Hello
root [4] .x say.C("Hi")
Hi
```

TFile and TTrees

- Very often, 'high level' HEP experimental data (i.e. the ones that analyst use to perform Physics analyses) are stored in ROOT 'TFiles'. In the file the data are organized by using a 'TTree' object.

Example: check the content of the root file.

- Open a file for reading:

```
root[] TFile f("Example.root")
```

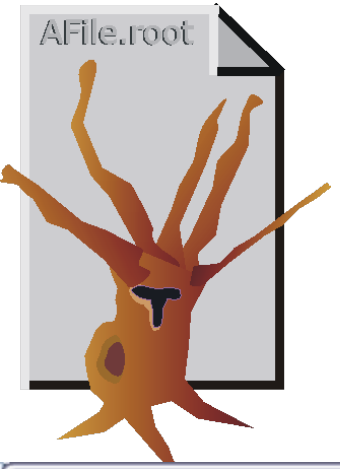
- Check file content:

```
root[] f.ls()
TFile**         Example.root    ROOT file
 TFile*         Example.root    ROOT file
  KEY: TTree    myTree;1        Example ROOT tree (more about this next slide)
  KEY: TH1F     totalHistogram;1        Total Distribution
  KEY: TH1F     mainHistogram;1 Main Contributor
  KEY: TH1F     s1Histogram;1   First Signal
  KEY: TH1F     s2Histogram;1   Second Signal
```

- Get a particular content by name

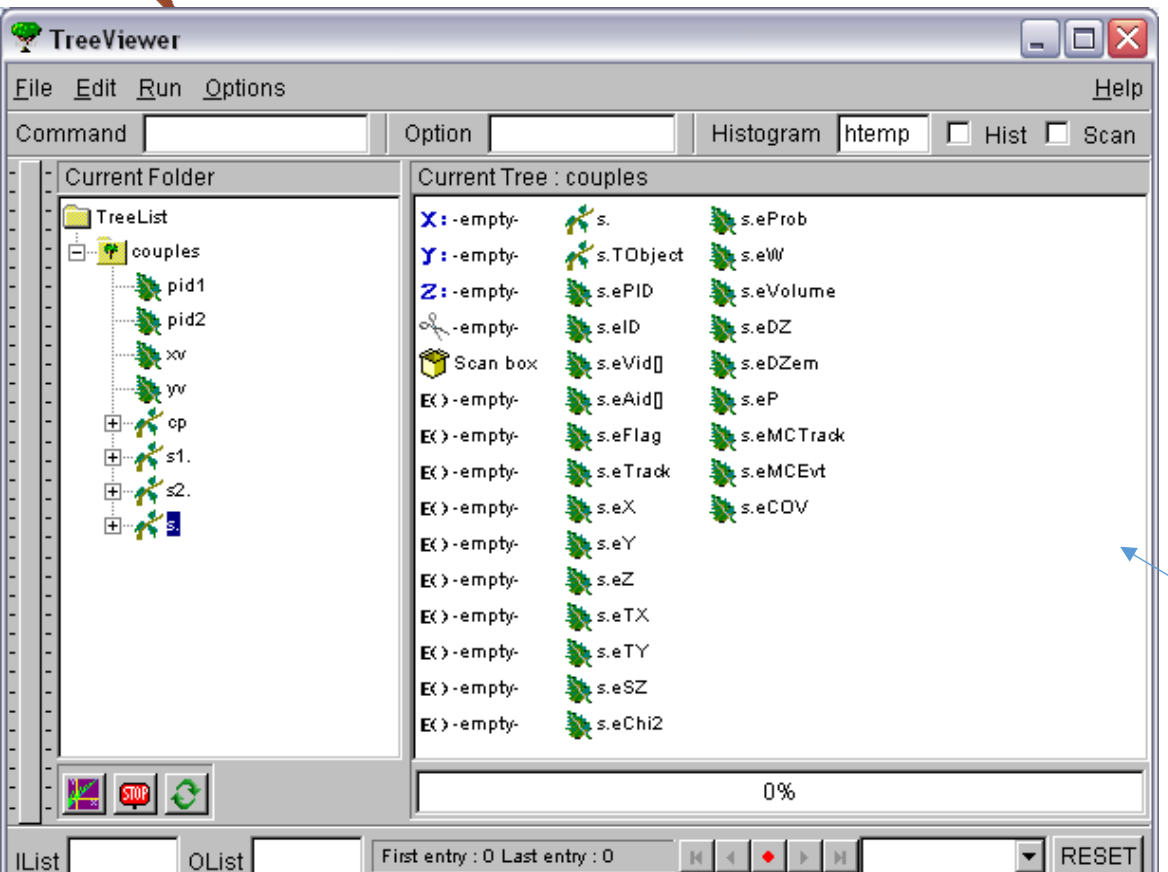
```
root[] totalHistogram->Draw();
root[] TH1F* myHisto = (TH1F*)
  f.Get("totalHistogram");
```

The ROOT TTree object



TTree is a ROOT object, can be seen as a very flexible container, a table where:

- In each row there is an ‘entry’ (typical HEP example is an ‘event’ or the detector status for a given trigger)
- In different columns there are different objects (whatever ROOT object you consider suitable to contain the data) that logically belong to the same events. Example: vector of reconstructed particles, DAQ event number, Timestamp ... can be in different columns.

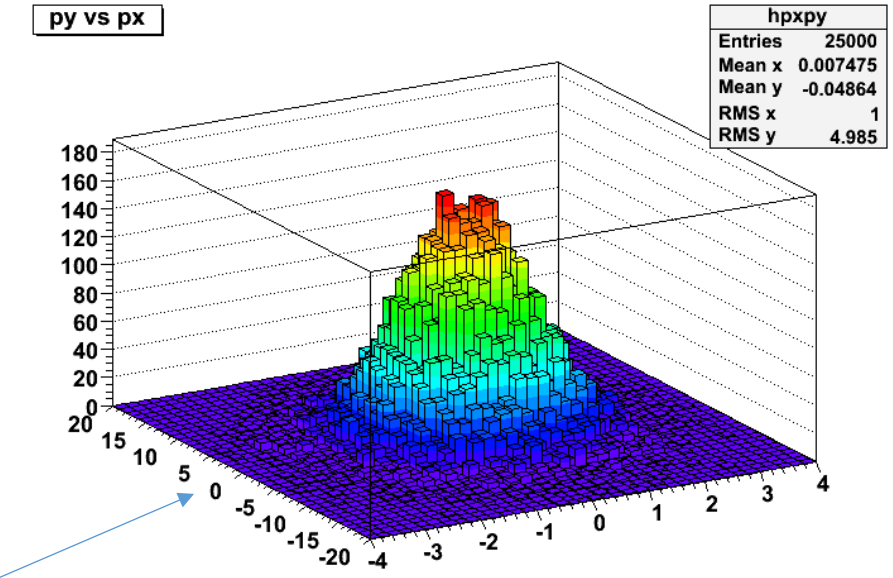


- TTree allows the storage of a very large amount of entries. A tree has a hierarchical structure with “Branch and Leaf”. This is useful if one want to read only the data belonging to that Branch/Leaf without loading in memory all the content of the Tree (fastest)

Interactive view of a Ttree:

```
[] TTree* myTree = (TTree*) f.Get("name");
[] myTree->StartViewer();
```

Useful commands for the interactive analysis of a TTree



Variable list (leafs and branches):

```
[ ]> tree->Print()
```

Plot 1D of a variable

```
[ ]> tree->Draw("varname")
```

Scatter plot of two variables:

```
[ ]> tree->Draw("varname1:varname2")
```

...with graphical option (lego2)

```
[ ]> tree->Draw("varname1:varname2", "", "lego2")
```

...with cut on a third variable:

```
[ ]> tree->Draw("varname1:varname2", "varname>3", "lego")
```

Scatter plot of three variables

```
[ ]> tree->Draw("varname1:varname2:varname3")
```

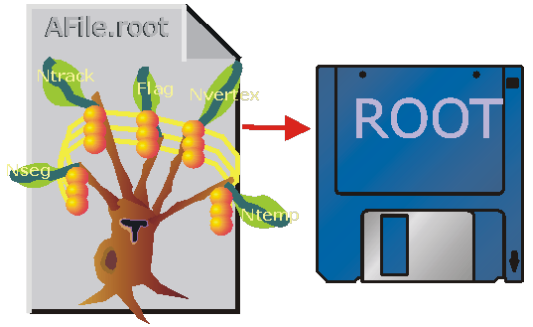
Usage of the class `TCut` for the definitions of the cuts to be used in the analysis

```
[ ]> TCut cut1="varname1>0.3"
[ ]> tree->Draw("varname1:varname2",cut1)
[ ]> TCut cut2="varname2<0.3*varname1+89"
[ ]> tree->Draw("varname1:varname2",cut1 && cut2)
```

Creating and filling a TTree in a root macro

```
TFile *myfile = new TFile("test.root","RECREATE");
TTree *tree = new TTree("myTree","A ROOT tree");

Now add a branch (class Event):
Event *event = new  Event();
myTree->Branch("EventBranch","Event",&event);

Add also a Tleaf:
Int_t ntrack; myTree->Branch("NTrack",&ntrack,"ntrack/I");
```

- To fill the Tree:
  - Assign the value to ntrack, event …
  - myTree->Fill();
- To save the Tree:
  - myTree->Write();



Read a TTree in a root macro

```
This is needed for more complex analysis, see $ROOTSYS/tutorials/tree/tree1.C

TFile f("tree1.root")
```

Get the Tree by name:

```
TTree * t1 =    (TTree*)f.FindObject("t1")
```

Create the appropriate variables to contain the data (the same as the slide before):

```
Int_t Ntrack_read;
```

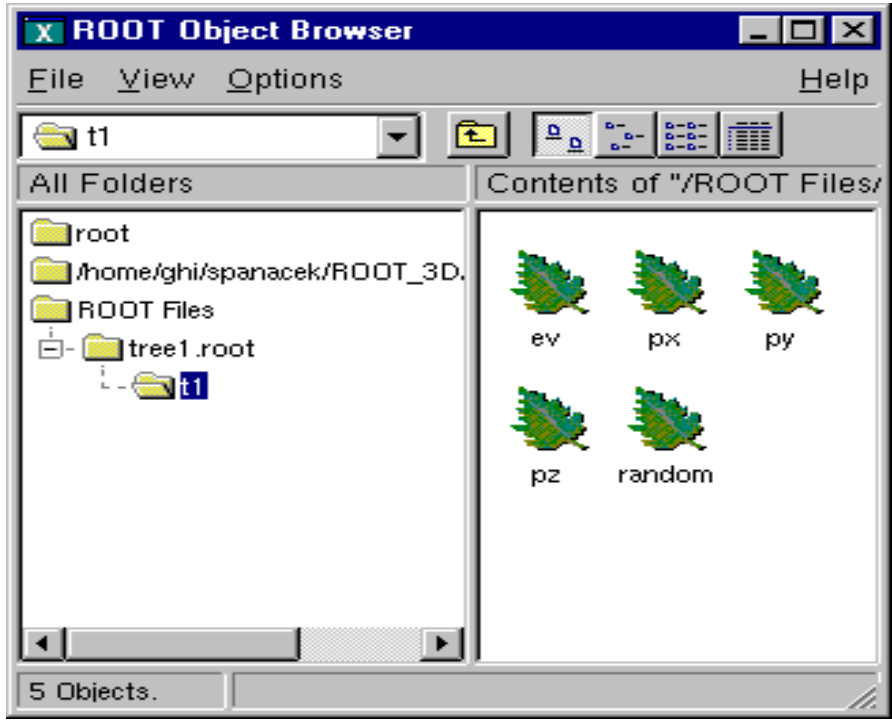
Associate the Branch/Leaf one wants to read to the variables by using the method:

```
SetBranchAddress("name", address)

    t1->SetBranchAddress("px NTrack", &Ntrack_read)
```

To read the TTree entry in position nr we can use the Ttree method

```
GetEntry(nr)

    t1->GetEntry(0) // first entry
```

Usually, one wants to loop over all the entries, from 0 to `t1->GetEntries()`



More details about fitting with root

Let's see here how to *define a generic function* to be used to fit a histogram. You can define like this all the functions you like that are writable in a C++ format, and include all the functions defined in the CERN math libraries. The code of 3 examples is shown below, you should include them in a root macro, in its global scope.

```
// Quadratic background function
Double_t background(Double_t *x, Double_t *par) {
   return par[0] + par[1]*x[0] + par[2]*x[0]*x[0];
}

// Lorentzian Peak function
Double_t lorentzianPeak(Double_t *x, Double_t *par) {
   return (0.5*par[0]*par[1]/TMath::Pi()) / TMath::Max(1.e-10,
   (x[0]-par[2])*(x[0]-par[2])+ .25*par[1]*par[1]);
}

// Sum of background and peak function
Double_t fitFunction(Double_t *x, Double_t *par) {
   return background(x,par) + lorentzianPeak(x,&par[3]);
}
```

Structure of the user defined function

This is a pointer to the parameters, it specifies the array of the parameters associated to the function.

This is the independent variable of this 1D function

```
// Quadratic background function
Double_t background(Double_t *x, Double_t *par) {
   return par[0] + par[1]*x[0] + par[2]*x[0]*x[0];
}

// Lorentzian Peak function
Double_t lorentzianPeak(Double_t *x, Double_t *par) {
   return (0.5*par[0]*par[1]/TMath::Pi()) / TMath::Max(1.e-10,
   (x[0]-par[2])*(x[0]-par[2])+ .25*par[1]*par[1]);
}

// Sum of background and peak function
Double_t fitFunction(Double_t *x, Double_t *par) {
   return background(x,par) + lorentzianPeak(x,&par[3]);
}
```

N.B.

- The range where the function is defined will be set in the calling part of the program, the user should check that the function is well defined in the specified range
- We will see how to set the maximum range of the fit parameters where ROOT will try to find the best solution.

Example on how to fit (user defined):

Here I just define an histogram and I fill it according to an array of data specified with the variable data[]

Here I define a root TF1 function and I tell ROOT that it is defined by my code specified outside the main part of the program.

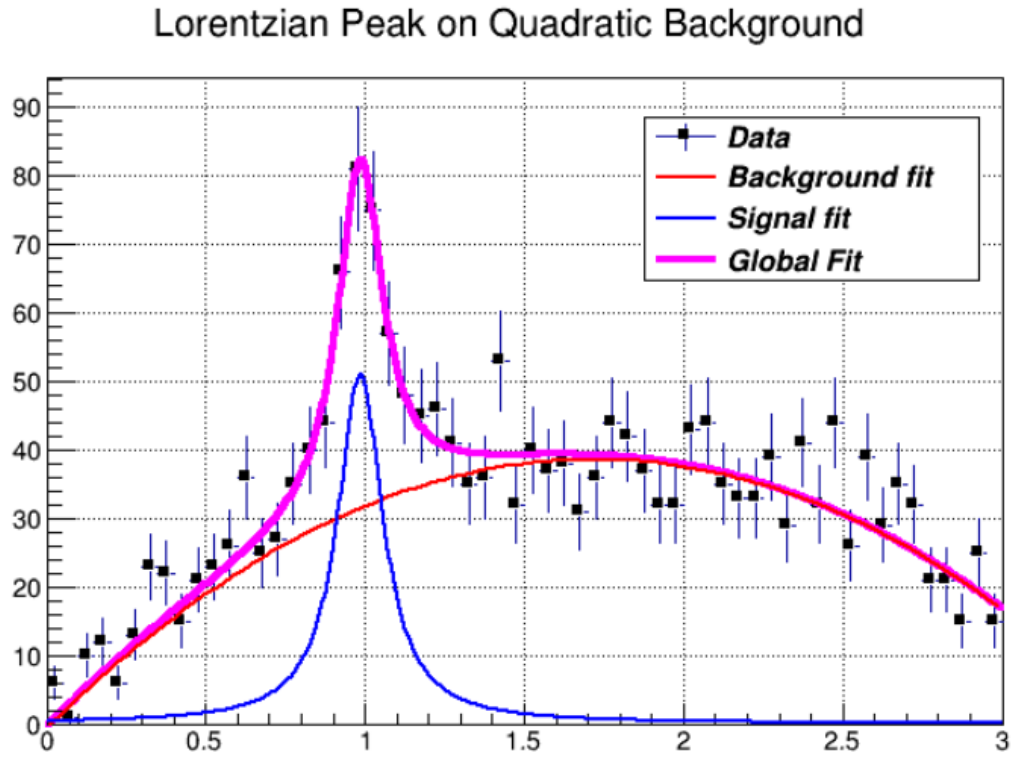
Notice:

The definition of the number of parameters (6)
The range where we want to perform the fit (0,3)

```
Double_t background(Double_t *x, Double_t *par) {
   return par[0] + par[1]*x[0] + par[2]*x[0]*x[0];
}

// Lorentzian Peak function
Double_t lorentzianPeak(Double_t *x, Double_t *par) {
   return (0.5*par[0]*par[1]/TMath::Pi()) / TMath::Max(1.e-10,
   (x[0]-par[2])*(x[0]-par[2])+ .25*par[1]*par[1]);
}

// Sum of background and peak function
Double_t fitFunction(Double_t *x, Double_t *par) {
   return background(x,par) + lorentzianPeak(x,&par[3]);
}

void FittingDemo() {
// bevington exercise by P. Malzacher, modified by R. Brun
const int nBins = 60;
Stat_t data[nBins] = {  6, 1,10,12, 6,13,23,22,15,21,
23,26,36,25,27,35,40,44,66,81,
75,57,48,45,46,41,35,36,53,32,
40,37,38,31,36,44,42,37,32,32,
43,44,35,33,33,39,29,41,32,44,
26,39,29,35,32,21,21,15,25,15};
TH1F *histo = new TH1F("example_9_1",
"Lorentzian Peak on Quadratic Background",60,0,3);

for(int i=0; i < nBins;  i++) {
   // we use these methods to explicitly set the content
   // and error instead of using the fill method.
   histo->SetBinContent(i+1,data[i]);
   histo->SetBinError(i+1,TMath::Sqrt(data[i]));
}
// create a TF1 with the range from 0 to 3 and 6 parameters
TF1 *fitFcn = new TF1("fitFcn",fitFunction,0,3,6);

// first try without starting values for the parameters
// this defaults to 1 for each param.
histo->Fit("fitFcn");
```

Extract the result of the fit

```
root[] TF1 *fit = hist->GetFunction(function_name);
root[] Double_t chi2 = fit->GetChisquare();
// value of the first parameter
root[] Double_t p1 = fit->GetParameter(0);
// error of the first parameter
root[] Double_t e1 = fit->GetParError(0);
```



or more in general

```
int fitStatus =  hist->Fit(myFunction);    // TFitResultPtr contains only the fit status

TFitResultPtr r = hist->Fit(myFunction,"S");  // TFitResultPtr contains the TFitResult
TMatrixDSym cov = r->GetCovarianceMatrix();   //  to access the covariance matrix
Double_t chi2   = r->Chi2();                  // to retrieve the fit chi2
Double_t par0   = r->Parameter(0);            // retrieve the value for the parameter 0
Double_t err0   = r->ParError(0);             // retrieve the error for the parameter 0
r->Print("V");                                // print full information of fit including covariance matrix
r->Write();                                   // store the result in a file
```

Useful fitting-related methods

How the root fit works

When you call the 'Fit' method, ROOT will ask the TMinuit program (or Minuit2, C++ version) to find the best parameters to fit the data. By default the Fit method will use an iterative program to find the minimum value of the χ^2 but other options can be used

Fitting Options

TH1::Fit(const char* fname, Option_t* option, Option_t* goption, Axis_t xmin, Axis_t xmax)

fname	Function name
option	Fitting options: "W" Set all errors to 1 "I" Use integral of function in bin instead of value at bin center "L" Use log likelihood method (default is chi-square) "LL" Use log likelihood method and bin contents are not integers "U" Use a user-specified minimization algorithm (via SetFCN) "Q" Quiet mode (minimum printing) "V" Verbose mode (default is between Q and V) "E" Perform better error estimation using MINOS technique "M" More. Improve fit results "R" Use the range specified in the function range "N" Do not store the graphics function. Do not draw. "0" Do not plot the result of the fit "+" Add this new fitted function to the list of fitted functions
goption	Graphical options
xmin - xmax	Fitting range

So far, we have been fitting by the chi-square method.

More details in:
https://root.cern/root/htmldoc/guides/users-guide/FittingHistograms.html , https://root.cern/manual/fitting

Reminder:

Fitting binned datasets: Chi-square vs. Log Likelihood

Chi-square

- Assumes that the events within each bin of the histogram are gaussianly distributed
- Works well at high statistics, but doesn't always give reliable results when some of the histogram bins have few (or zero) entries.

Binned Log Likelihood

- Correct Poisson treatment of low statistics bins (≲10 events)
- Recommended when some histogram bins have few (or zero) entries

Given a set of parameters, what is the probability that your real dataset of "y_i's" could occur?

Gaussian case:

$$P(a_0 \ldots a_k) = \prod_{i=1}^{nbins} \frac{1}{\sigma_i \sqrt{2\pi}} e^{-\frac{1}{2}\sum \frac{(y_i - f(x_i; a_0 \ldots a_k))^2}{\sigma_i^2}}$$

$$\chi^2(a_0 \ldots a_k) = \sum_{i=1}^{nbins} \left(\frac{y_i - f(x_i; a_0 \ldots a_k)}{\sigma_i} \right)^2$$

Maximizing this probability is equivalent to minimizing the sum in the exponential... which we call χ^2

Poissonian case:

$$P(a_0 \ldots a_k) = \prod_{i=1}^{nbins} \frac{[f(x_i; a_0 \ldots a_k)]^{y_i}}{y_i!} e^{-f(x_i; a_0 \ldots a_k)}$$

$$\ln P(a_0 \ldots a_k) = \sum [y_i \ln f(x_i; a_0 \ldots a_k)] - \sum f(x_i; a_0 \ldots a_k) + const$$

It is easier (and equivalent) to maximize the natural log of the probability (or to minimize the negative log)

Message:

It is a good idea to study the dependence of your fit results from the fit options

- Binned Maximum Likelihood in between $-\ln L(p)_{\text{binned}} = \sum_{\text{bins}} n_{\text{bin}} \ln F(\vec{x}_{\text{bin-center}}, \vec{p})$
  - Much faster than full Maximum Likihood
  - Correct Poisson treatment of low statistics bins
  - Misses information with feature size < bin size

General usage of TMinuit

Root and TMinuit can be used, for generic parameter estimation problems (outside the Fit function).

Indeed MINUIT is a program to calculate numerically:

- a function minimum of $F(\vec{a})$, of (max. 50) parameters a_i
- the covariance matrix of these parameters
- the (asymmetric or parabolic) errors of the parameters from $F_{\min} + \Delta$ for arbitrary Δ
- the contours of parameter pairs a_i, a_j

The user has to define a function to be minimized.

Example: A set of rate measurements at fixed intervals of a radioactive source yielded:

$$r_i = [1, 1, 5, 4, 2, 0, 3, 2, 4, 1, 2, 1, 1, 0, 1, 1, 2, 1]$$

Find the mean with Minuit. Of course this problem can be solved without Minuit. But imagine that you have data with different errors given by third measurements etc.. There the usage of Minuit can make your life easier.

General usage of TMinuit

Data and definition of the funtion to be minimized

```
#define NDATA 18
int r[NDATA] = {1,1, 5, 4,2,0,3,2, 4,1,2,1,1,0,1,1,2,1};
int rfac[NDATA] = {1,1,120,24,2,1,6,2,24,1,2,1,1,1,1,1,2,1 };
```

```
void fcn(int &npar, double *gin, double &f, double *par,
    int iflag)
{
   int i;
   double mu, lnL;

   mu = par[0];
   lnL = 0.0;
   for(i=0;i<NDATA; i++)
    {
     lnL += r[i]*log(mu) - mu - log((double) rfac[i]);
    }
   f = -lnL;
}
```

Initialization

```
main()
{
   double arglist[10];
   int ierflg = 0;

   double start   = 1.0;
   double step = 0.1;
   double l_bnd = 0.1;
   double u_bnd = 10.;

   TMinuit minuit(1);
   minuit.SetFCN(fcn);
   minuit.mnparm(0,"Poisson mu", start, step, l_bnd,
    u_bnd, ierflg);
}
```

Parameter definition

Execution

defines function value above minimum value for error and contour calculation

```
arglist[0] = 0.5;
minuit.mnexcm("SET ERR",arglist,1,ierflg);

minuit.mnexcm("MIGRAD",arglist,0,ierflg);
```

https://root.cern/manual/math/#tminuit for more, including Minuit2

ROOT for Unfolding

Several algorithms have been written already in root to solve unfolding problem:

- TUnfold is based on a least square fit with Tikhonov regularisation

- User has to provide data histograms, response matrix, measurement covariance matrix.

TUnfold solves the inverse problem

```
  chi**2 = 1/2 * (y-Ax)# V (y-Ax) + tau (L(x-x0))# L(x-x0)

where # means that the matrix is transposed

Monte Carlo input

  y: vector of measured quantities  (dimension ny)
  V: inverse of covariance matrix for y (dimension ny x ny)
     in many cases V is diagonal and calculated from the errors of y
  A: migration matrix                (dimension ny x nx)
  x: unknown underlying distribution (dimension nx)

Regularisation

  tau: parameter, defining the regularisation strength
  L: matrix of regularisation conditions (dimension nl x nx)
  x0: bias distribution

and chi**2 is minimized as a function of x

This applies to a very large number of problems, where the measured
distribution y is a linear superposition of several Monte Carlo shapes
and the sum of these shapes gives the output distribution x
```

ROOT for Unfolding

RooUnfold is a package containing many unfolding algorithms, TUnfold is one of them

- The Tunfold algorithm is used in this example where we want to unfold the data containing two Breight wigner peaks. Data are affected by a Gaussian smearing.
- The program return the unfolded distribution the correlation matrix, the $\chi 2$, the bias...
- Have a look to https://root.cern.ch/doc/master/classTUnfold.html for more details, including improved https://root.cern.ch/doc/master/classTUnfoldDensity.html

QUESTIONS?

SECOND PART:

TMVA, shortly

About TMVA:

The Toolkit for MultiVariate Analysis
(TMVA) is a library that allows to create test statistics for classification from data belonging to known categories (eg. Monte Carlo) and to classify unknown data using these test statistics. TMVA supports several methods, including artifical neural networks (ANN), boosted decision trees (BDT) and the Fisher Linear Discriminant Analysis method (Fisher LDA), which we are already familiar with from the homework problems. The TMVA library is included in ROOT since 2013.

Example:

We need some data to classify. Let's download data about Italian wines, from the same region, but produced from different grape cultivars. This data is available from https://web.archive.org/web/20220320033501/https://archive.ics.uci.edu/ml/machine-learning-databases/wine/ [1]. Also get the User's Guide from https://github.com/root-project/root/blob/master/documentation/tmva/UsersGuide/TMVAUsersGuide.pdf. Each line in the file has 14 numbers: grape cultivar, alcohol content, malic acid, ash, alcalinity of ash, magnesium, total phenols, flavanoids, nonflavanoid phenols, proanthocyanins, color intensity, hue, OD280/OD315 of diluted wines and proline.

We can use an artificial neural network to classify the wines. For a good article about artificial neural networks, see [2]. The code below trains the network and classifies the wines. This code uses a MultiLayer Perceptron Artificial Neural Network (MLP_ANN), with a hidden layer of 10 neurons and a boosted decision tree (BDT). The same code can be easily changed to use other methods.

See code Wine.cc:

TMVA References:

[1] A. Asuncion and D.J. Newman, UCI machine learning repository, 2007.
[2] M.W Gardner and S.R Dorling, Artificial neural networks (the multilayer perceptron) - a review of applications in the atmospheric sciences, Atmospheric Environment 32 (1998), no. 14-15, 2627 – 2636.
[3] https://root.cern/manual/tmva/

QUESTIONS?

Credits:

- Mirko Berretti, Jan Welti
- Luciano Pandola (based on S. Panacek & N. Di Marco) https://agenda.infn.it/getFile.py/access?sessionId=8&resId=3&materialId=0&confId=3085
- **Sungkyun Park, http://hadron.physics.fsu.edu/~skpark/root.html**
- https://root.cern.ch/root/htmldoc/guides/users-guide/FittingHistograms.html#creating-user-defined-functions-tf1
- Jen Raaf 2011 REU Root Tutorial @ Duke, http://hep.bu.edu/~jlraaf/2011REU/root_lecture02.pdf
- Wouter Verkerke, UCSB https://indico.nbi.ku.dk/getFile.py/access?contribId=18&resId=0&materialId=slides&confId=100
- Christoph Rosemann, http://www.desy.de/~rosem/flc_statistics/data/04_parameters_estimation-C.pdf