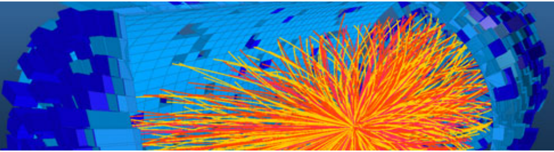
ROOT
Data Analysis Framework



Introduction to ROOT and application to data analysis at the LHC

Nguyen Thi Hong Van

INSTITUTE OF PHYSICS, HANOI

August 13, 2014

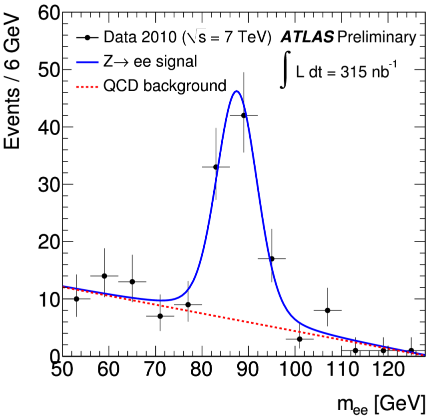
Outline

1. ROOT: Motivation and Introduction
2. ROOT basics
3. ROOT analysis
4. Application to data analysis at the LHC

Motivation for using ROOT

Tasks in experimental physics:

- Comparisons between experiments and theoretical models
- $\Rightarrow$ A visualization of data is thus needed
- $\Rightarrow$ A powerful library of mathematical functions and procedures is needed
- Handling errors of measurements properly
- Fit function and means to quantify the level of agreement between observation and model
- Deal with a huge of data volume
- Simulation: create pseudo-data to estimate errors of experiments or to test the effects of different assumptions



ROOT: Introduction (1/2)

What is ROOT?

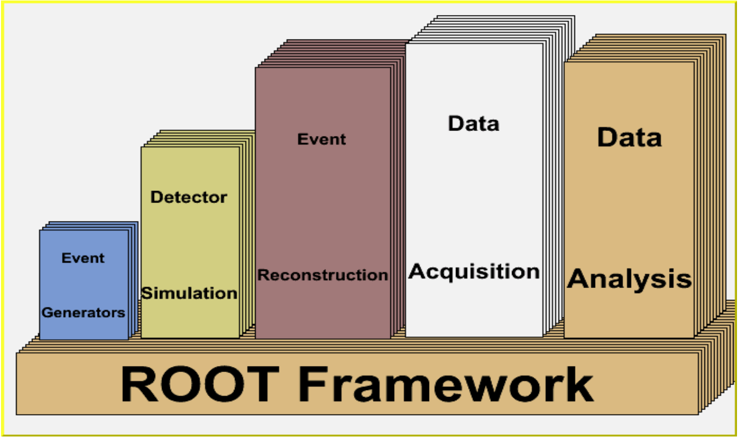
- ROOT is an object-oriented C++ analysis package
- ROOT is born at CERN (European Centre for Particle Physics)
- ROOT uses a language called CINT (C/C++ Interpreter) which contains several extensions to C++

ROOT is a powerful software framework

- **Save data**: ROOT provides a data structure that is extremely powerful for fast access of huge amounts of data
- **Access data**: data saved into one or several ROOT files can be accessed from your PC, from the web and from large-scale file delivery systems used e.g. in the GRID.
- **Process data**: powerful mathematical and statistical tools are provided to operate on your data.
- **Show results**: Results are best shown with histograms, scatter plots, fitting functions, etc. ROOT graphics may be adjusted real-time by few mouse clicks. High-quality plots can be saved in PDF or other format.

Every day, thousands of physicists use ROOT applications for data analysis, etc.

ROOT: Introduction (2/2)



ROOT installation

Some useful links

- ROOT home page: http://root.cern.ch/
- Downloading ROOT: http://root.cern.ch/drupal/content/downloading-root
- ROOT tutorials: http://root.cern.ch/drupal/content/tutorials

Install ROOT

- For Linux and Mac OS:
  - download ROOT from source:
  - extract the tar file to folder root
  - at root do ./configure then make
- For Window:

ROOT Basic

Commands in ROOT

- Start ROOT: type " root " or " root -l "
- Syntax for a command: root [1] .<command>
- Quit ROOT: type .q
- Obtain a list of commands: .? or .h
- Access the shell of the operating system: .!<OS_command>
- Execute a macro: .x <file_name >

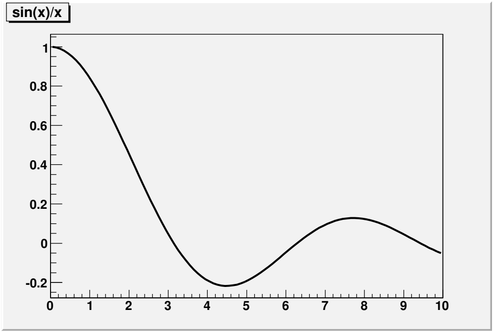
ROOT as a calculator: TMath

```
root [0] 1+2
(const int) 3
root [ 1 ] TMath::Pi()
( Double_t ) 3.14159265358979312e+00
root [1] TMath::Sin(TMath::Pi()/2)
(Double_t)1.00000000000000000e+00
```

ROOT basic

Plotting a function in ROOT

```
root [3] TF1 *f1 = new TF1 ( " f1 " , " sin (x)/x" , 0 . , 1 0 . ) ;
root [4] f1 → Draw ( ) ;
<TCanvas::MakeDefCanvas>: created default TCanvas with name c1
root [5] TF1 *f2 = new TF1 ( " f2 " , " [0]* sin ([1]*x)/x" , 0 . , 1 0 . ) ;
root[6] f2→SetParameters ( 1 , 1 ) ;
root[7] f2→Draw();
```



ROOT basic: Graphs in ROOT

Graphs in ROOT

- A graph is a graphics object made of two arrays X and Y, holding the x,y coordinates of n points
- Graph classes: **TGraph**, **TGraphErrors**, **TGraphAsymmErrors**, and **TMultiGraph**.

TGraphErrors example: plotting experiments

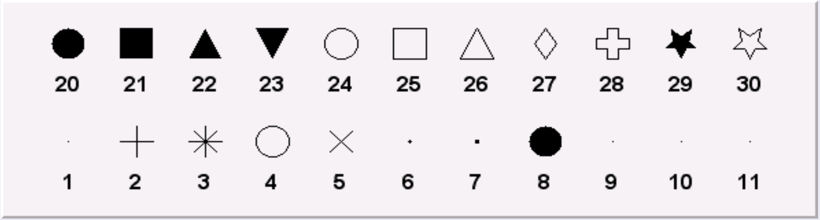
- root [5] Int_t n = 10;
- root [6] Float_t x[n] = { -0.1 , .2 , .5 , .6 , .8 , .9 , 1.2 , 1.5 , 2 , 2.4 };
- root [7] Float_t y[n] = { .5 , 2 , 3 , 4.5 , 6 , 8 , 9.5 , 7 , 5.5 , 3 };
- root [8] Float_t ex[n] = { .04 , .05 , .07 , .03 , .1 , .07 , .1 , .05 , .03 , .1 };
- root [9] Float_t ey[n] = { .09 , 0.1 , .3 , .5 , .2 , .4 , .5 , .8 , .4 , .6 };
- root [10] gr = new TGraphErrors(n , x , y , ex , ey);
- root [11] gr→SetTitle("TGraphErrors Example");
- root [12] gr→SetMarkerColor(4);
- root [13] gr→SetMarkerStyle(21);
- root [14] gr→Draw("ALP");

ROOT Basics: drawing options for graphs

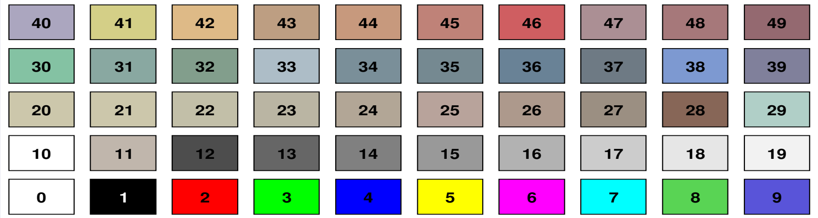
graphs' plotting options

- " A ": axis are drawn around the graph
- " L": a simple polyline is drawn
- "F ": a fill area is drawn ('CF' draw a smoothed fill area)
- " C ": a smooth Curve is drawn
- " * ": a Star is plotted at each point
- " P ": the current marker is plotted at each point
- " B ": a Bar chart is drawn
- " 1 ": "1" when a graph is drawn as a bar chart, this option makes the bars start from the bottom of the pad. By default they start at 0
- " X+ ": the X-axis is drawn on the top side of the plot
- " Y+ ": the Y-axis is drawn on the right side of the plot.

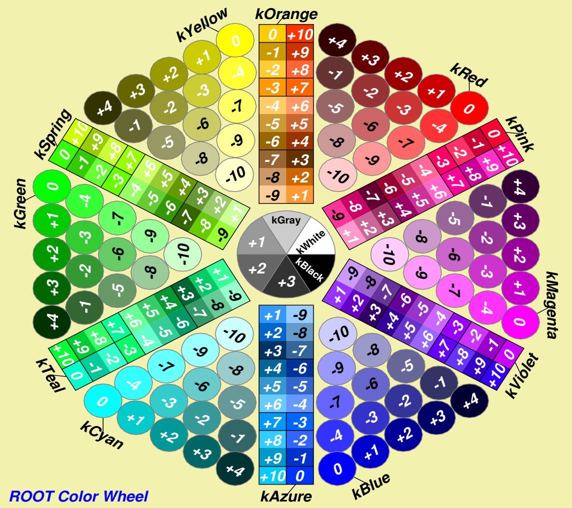
Marker styles



Basic colors



Color wheel


kYellow
kOrange
kRed
kPink
kMagenta
kViolet
kBlue
kAzure
kCyan
kTeal
kGreen
kSpring
kGray
kWhite
kBlack
ROOT Color Wheel

ROOT basics: more on drawing options

Latex in ROOT

♣	#club	♦	#diamond	♥	#heart	♠	#spade
℘	#voidn	ℵ	#aleph	ℑ	#Jgothic	ℜ	#Rgothic
≤	#leq	≥	#geq	〈	#LT	〉	#GT
≈	#approx	≠	#neq	≡	#equiv	∝	#propto
∈	#in	∉	#notin	⊂	#subset	⊄	#notsubset
⊃	#supset	⊆	#subseteq	⊇	#supseteq	∅	#oslash
∩	#cap	∪	#cup	∧	#wedge	∨	#vee
©	#ocopyright	©	#copyright	®	#oright	®	#void1
™	#trademark	™	#void3	Å	#AA	å	#aa
×	#times	÷	#divide	±	#pm	⁄	#/
•	#bullet	°	#circ	⋯	#3dots	·	#upoint
ƒ	#voidb	∞	#infty	∇	#nabla	∂	#partial
”	#doublequote	∠	#angle	↵	#downleftarrow	¬	#corner
∣	#lbar	∣	#cbar	—	#topbar	{	#ltbar
⎩	#arcbottom	⎧	#arctop	⌠	#arcbar	⌊	#bottombar
↓	#downarrow	←	#leftarrow	↑	#uparrow	→	#rightarrow
↔	#leftrightarrow	⊗	#otimes	⊕	#oplus	√	#surd
⇓	#Downarrow	⇐	#Leftarrow	⇑	#Uparrow	⇒	#Rightarrow
⇔	#Leftrightarrow	∏	#prod	∑	#sum	∫	#int
∣	#void8	□	#Box	⊥	#perp	⊙	#odot
ħ	#hbar	∥	#parallel				

ROOT basics: Histograms in ROOT

Histogram declaration

- Histograms can be 1-D (class " TH1F "), 2-D (class " TH2F ") or 3-D (class " TH3F ")
- Declare a histogram to be filled with floating point numbers:

  `TH1F *histName = new TH1F("histName", "histTitle", n_bins,x_low,x_high)`
- Example:

  `TH1F *h=new TH1F ( "h" , " example histogram " , 1 0 0 , 0 . , 5 . ) ;`
- Note:
  - Bin 0 → underflow (i.e. entries with x < x_low)
  - Bin (n_bins+1) → overflow (i.e. entries with x > x_high)
- 2-D and 3-D histograms can be booked similarly:

  `TH2F *h=new TH2F ( "h" , " my histo " , 1 0 0 , 0 . , 5 ., 200, 2., 40. ) ;`

Plotting histograms

- Fill a histogram: `histo→Fill(x_val);`
- Draw: `histo→Draw();`

Some basics in statistic

- Standard deviation of a distribution:

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2}$$

- Standard deviation of the mean:

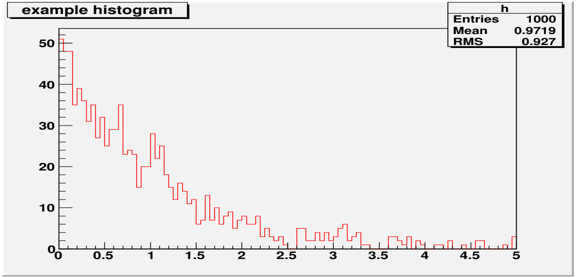
$$\sigma_{mean} = \frac{1}{\sqrt{N}}\sigma$$

- Confidence Interval:
    - $\sigma \rightarrow \mathrm{CI} = 0.6826895$
    - $2\sigma \rightarrow \mathrm{CI} = 0.9544997$
    - $3\sigma \rightarrow \mathrm{CI} = 0.9973002$
    - $4\sigma \rightarrow \mathrm{CI} = 0.9999366$
    - $5\sigma \rightarrow \mathrm{CI} = 0.9999994$

ROOT basics: Histograms in ROOT

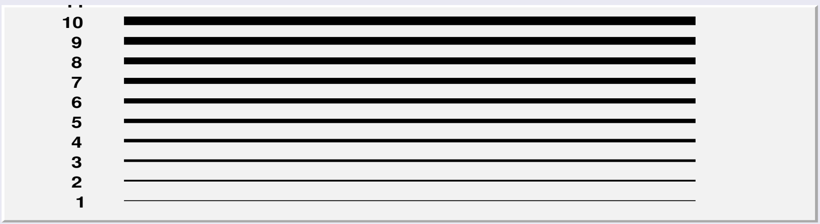
Example of 1D histogram

- root [0] TF1 efunc (" efunc " , " exp ([0]+[1]* x)" , 0 . , 5 .) ;
- root [1] efunc . SetParameter(1, -1)
- root [2] TH1F *h=new TH1F ("h" , " example histogram " , 1 0 0 , 0 . , 5 .) ;
- root [3] for (int i =0;i <1000; i++) {h→Fill (efunc.GetRandom ( )) ; }
- root [4] h→SetLineColor(2)
- root [5] h→Draw()
  <TCanvas::MakeDefCanvas>: created default TCanvas with name c1

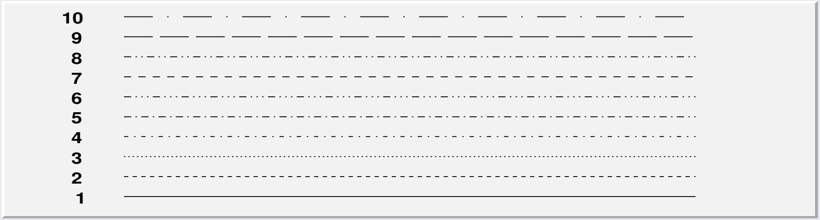


Drawing options

Line width: "histo→SetLineWidth()"; "histo→GetLineWidth()"

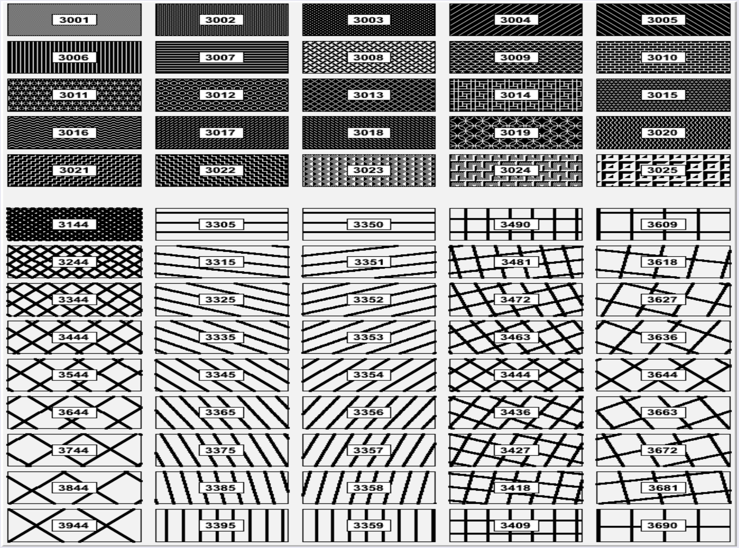


Line Style: "histo→SetLineStyle()"; "histo→GetLineStyle()"



Drawing options

Filled area style : “histo→SetFillStyle()”; “histo→GetFillStyle()”



ROOT basics: Drawing options for histograms

- " E " → Draw error bars.
- " AXIS " → Draw only axis.
- "AXIG " → Draw only grid (if the grid is requested).
- " HIST " → Visualize the histograms without error bars.
- " FUNC " → Draw the fit result only
- " SAME " → Superimpose on previous picture in the same pad.
- " LEGO " → Draw a lego plot with hidden line removal.
- " TEXT " → Draw bin contents as text
- " X+ " → The X-axis is drawn on the top side of the plot.
- " Y+ " → The Y-axis is drawn on the right side of the plot.
- " BOX " → A box is drawn for each cell with surface proportional to the content's absolute value : for 2D histograms
- " COL " and " COLZ " → A box is drawn for each cell with a color scale varying with contents: for 2D histogram
- " CONT " → Draw a contour plot: for 2D histogram

ROOT basics: more on 1D histogram and 2D histogram

1D histogram

- Create a gaussian histogram
- histo→GetMean() → Return mean value
- histo→GetMeanError(int axis=1) →Return standard error of mean of this histogram along the X axis
- histo→GetRMS(int axis=1) → For axis = 1,2 or 3 returns the Sigma value of the histogram along X, Y or Z axis
- histo→GetRMSError() → Return error of RMS estimation

2D histogram

- Create a 2D histogram
  TH2F* h2 = new TH2F("h2","", 50, -5, 5, 100, -10, 10)
- using gRandom→Rannor(x, y) → Return 2 numbers distributed following a gaussian with mean=0 and sigma=1.
- h2→ProjectionX("hx",-1,-1) ; h2→ProjectionY("hy",-1,-1)

ROOT basics: Interactive ROOT

- When the mouse is over an object, a right-click opens a pull-down menu displaying in the top line the name of the ROOT class you are dealing with
  - **TCanvas** for the display window itself,
  - **TFrame** for the frame of the plot,
  - **TAxis** for the axes,
  - **TPaveText** for the plot name,
  - **TF1** , **TH1F** and **TGraphErrors**
  - Try: TF1 with "SetLineAttributes" then " Set Parameters "; TH1F with " FitPanel "
- Drawing: eg. Feynman diagram...
- To save plots: $\rightarrow$SaveAs
- Note: a plot can be stored as a macro written in C++ language

ROOT basics: TLorentzVector

- Class TLorentzVector
- TLorentzVector fourVector(X, Y, Z, T)
- TLorentzVector fourMomentum(Px, Py, Pz, E)
- fourMomentum.M() → Return mass
- fourMomentum.Pt() → Return transverse momentum
- Summing two four-vector:
  TLorentzVector fourVectorTotal = fourVector1 + fourVector2
- fourVectorTotal.M() → Invariant mass of two particles
- fourVectorTotal.DeltaPhi()

ROOT analysis

- So far we have been working with ROOT prompt...
- Now we move to work with ROOT macros excuted by C++ interpreter (CINT)
- Create a file MacroExample.cxx

MacroExample.cxx

```
void Example() {
                    .....
          your lines of CINT code
                    ....
}
```

- The macro is executed by typing
  > root MacroExample.cxx
- Or it can be loaded and excuted in ROOT prompt
  root [0] . L MacroExample.cxx
  root [1] Example ( )

ROOT analysis: prepare for plotting

Re - initialise ROOT

- `gROOT → Reset ( ) ;` → re - initialise ROOT
- `gROOT → SetStyle ( " Plain " ) ;` → set empty TStyle
- `gStyle→SetOptStat ( 1 1 1 1 1 1 ) ;` → print statistics on plots, (0) for no output
- `gStyle→SetOptFit ( 1 1 1 1 ) ;` → print fit results on plot, (0) for no ouput
- `gStyle→SetOptTitle ( 0 ) ;` → suppress title box
- ...

Create a canvas

- `TCanvas c1 ( " c1 " , " <Title >" , 0 , 0 , 400 , 300 ) ;`
- `c1 . Divide ( 2 , 2 ) ;` → set subdivisions, called "pads"
- `c1 . cd ( 1 ) ;` → change to pad1 of canvas c1

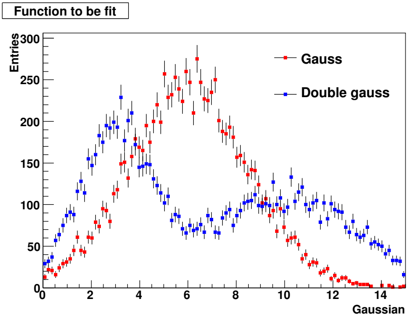
ROOT analysis: saving and accessing data

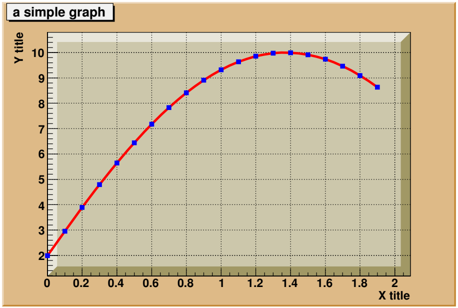
TFile

- Loading and looking at contents of a ROOT file "filename.root" :
  `> root -l filename.root`
  `root [1].ls`
  `root [2] TBrowser a`
- `TFile file ("filename.root");` →open an existing file (read only):
- `TFile file ("filename.root", "RECREATE");` → create a new file, if the file already exists it will be overwritten.
- `TFile file("filename.root", "UPDATE");` → open an existing file for writing. if no file exists, it is created.
- `TH1F* h = (TH1F*)file->Get("filename");` → get the histogram out of the file
- `histo->Write();` → write a histo to the file
- `file->Write(); file->Close();` → to save what you have done...

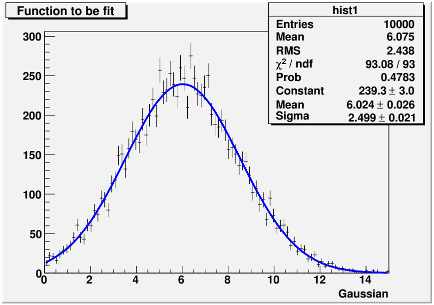
ROOT analysis: examples

- Example 1: write a maco to create and draw a histogram as shown in part 1
- Example 2: write a macro to create two histograms and write them to a ROOT file
- Example 3: write a macro to get two histograms from the above ROOT file then draw them; add a legend ...
- Example 4: write a macro to draw a simple graph





ROOT analysis: Fitting in ROOT



- TH1::Fit(char * fname, "options", x_min, x_max)
- TH1::GetFunction("fitFunction") → Obtain the fitted TF1 function
- TF1::GetNpar() → Get number of parameters
- TF1::SetParameter(parNo,value); TF1::SetParameters(val1,val2,...) → Set fit parameters
- TF1::GetParameter(parNo) → Get fit parameters
- TF1::GetChisquare() → Get χ^2 of fit
- TF1::GetNDF() → Get Number of Degrees of Freedom

ROOT analysis: TTree

- With **TTree** we can store large quantities of same-class objects
- **TTree** class is optimized to reduce disk space and enhance access speed
- **TTree** can hold all kind of data
- Examples: create a Tree and try to plot Tree variables ....

Some steps with Tree

- T→Print(); → Prints the content of the tree
- T→Scan(); → Scans the rows and columns
- T→Draw("x"); → Draw a branch of Tree
- T→Draw("x","x>0"); → Draw "x " when x>0
- T→Draw("x","x>0 && y>0"); → Draw " x " when x >0 and y >0
- T>Draw(" y","x ","same"); → Superimpose " y " on " x "
- T>Draw("y:x"); → Make "y versus x" 2-D plot
- T>Draw("z:y:x"); → Make 3-D plot
- T>Draw("sqrt(x*x+y*y)"); → Plot calculated quantity

References

- http://public.web.cern.ch/public/en/LHC/LHC-en.html
- http://www.atlas.ch/photos/index.html
- http://cms.web.cern.ch/
- http://root.cern.ch/drupal/content/tutorials-and-courses
- http://www.nevis.columbia.edu/ seligman/root-class/
- www-ekp.physik.uni-karlsruhe.de/ quast/Skripte/diving_into_ROOT.pdf