ROOT - Einleitung

Der Vergleich von Messungen mit theroretischen Modellen ist ein wichtiger Bestandteil der Experimentalphysik. Dazu müssen in der Regel Daten selektiert, statistisch analysiert, modelliert, Modelparameter angepasst und Fehler abgeschätzt werden.

- Mit den bisher gelernten C++ Elementen könnten wir im Prinzip die Daten eines Experimentes analysieren, aber es fehlt z.B. noch
  - graphische Darstellung
  - Datenanpassung
  - I/O für komplexere Datenstrukturen (HEP Experimente > 10^6 Kanäle)
  - Speichern von Objekten

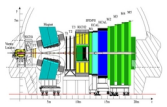
ROOT Data Analysis Framework entwickelt seit 1995 am CERN als Open Source Project unter GNU LGPL zur Datenanalyse am LHC als C++ Klassenbibliothek.

- 10 Vollzeit-Entwickler am CERN
- Offener Quellcode → software wird von den Usern mit- und weiterentwickelt
- Dokumentation, Tutorials und Quellcode: https://root.cern.ch/

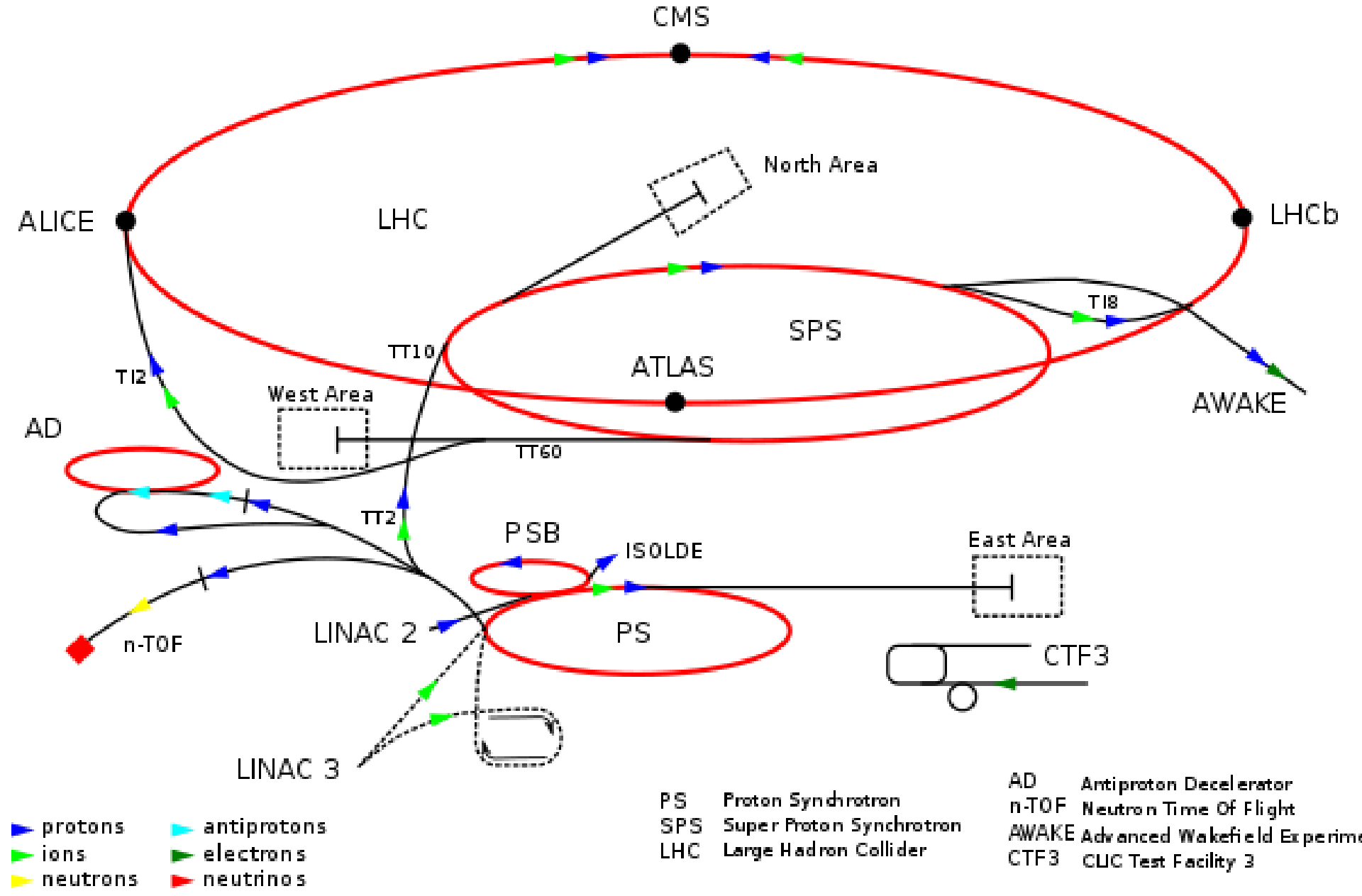
Large Hadron Collider at CERN

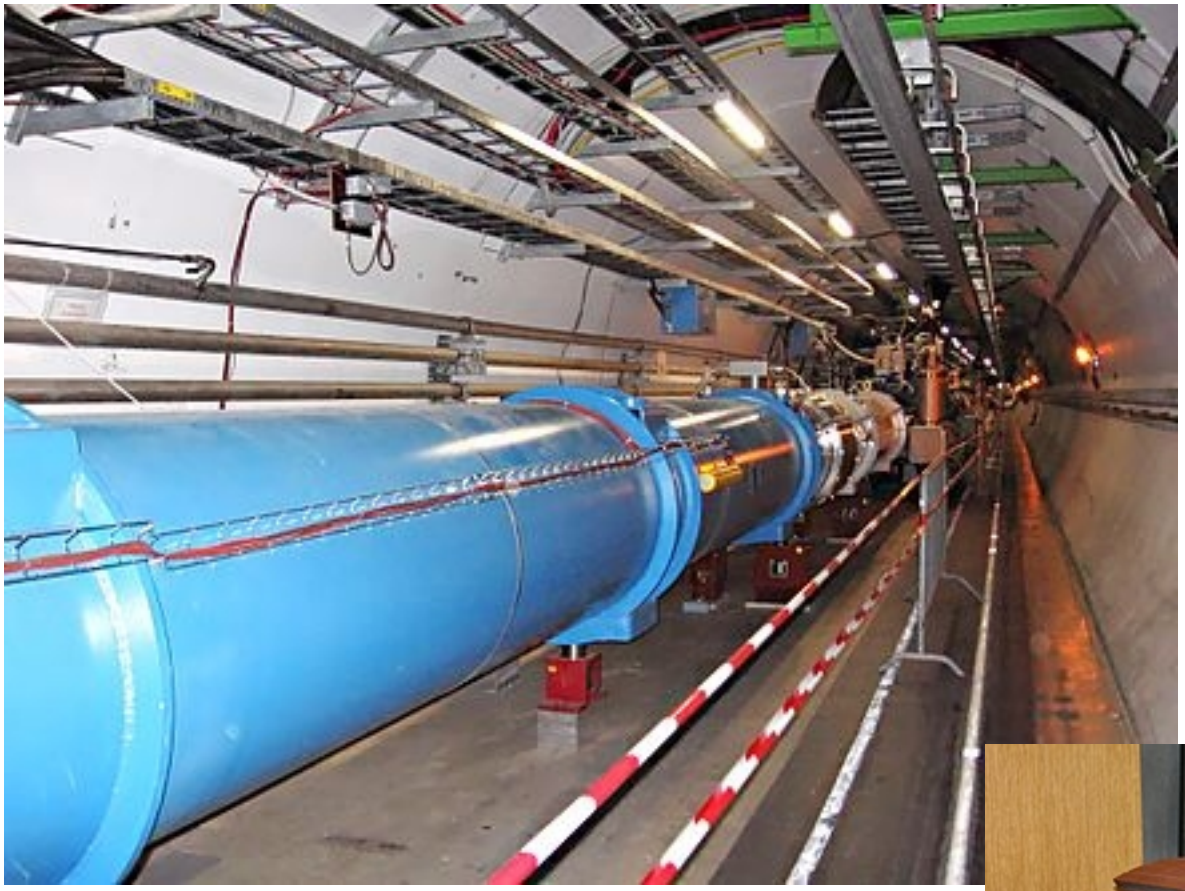
CERN – in a Nutshell

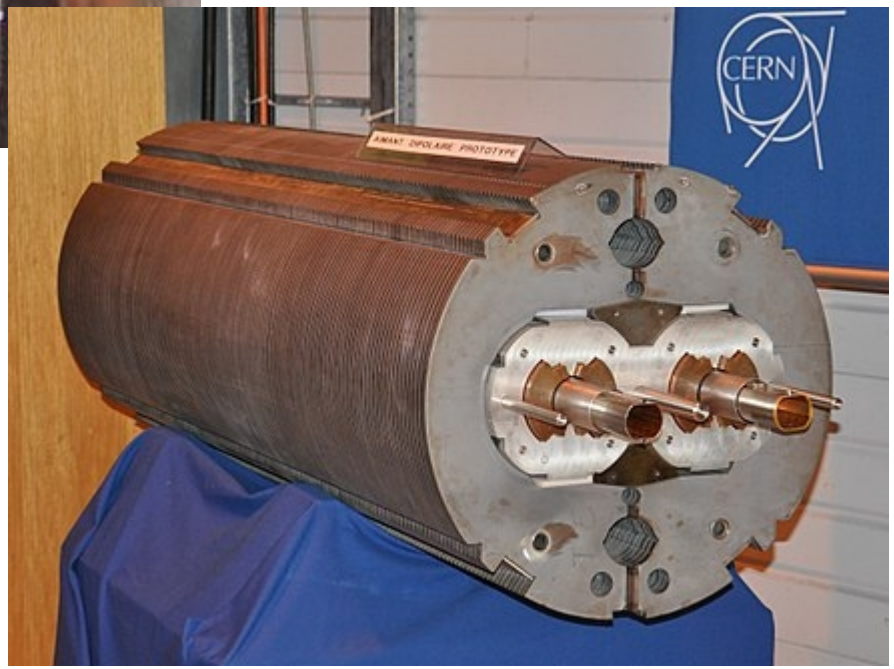
- CERN = Conseil Europeen pour la Recherche Nucleaire
- Is the largest research facility for particle physics world wide
- Is located in Switzerland, in Meyrin close by Geneva
- It has 24 member states from Europe with others associated
- ~ 3300 employees and about 14000 guest scientists
- The annual budget is about 1230 million Euro
- The facility has 3 accelerators, PS, SPS and the LHC

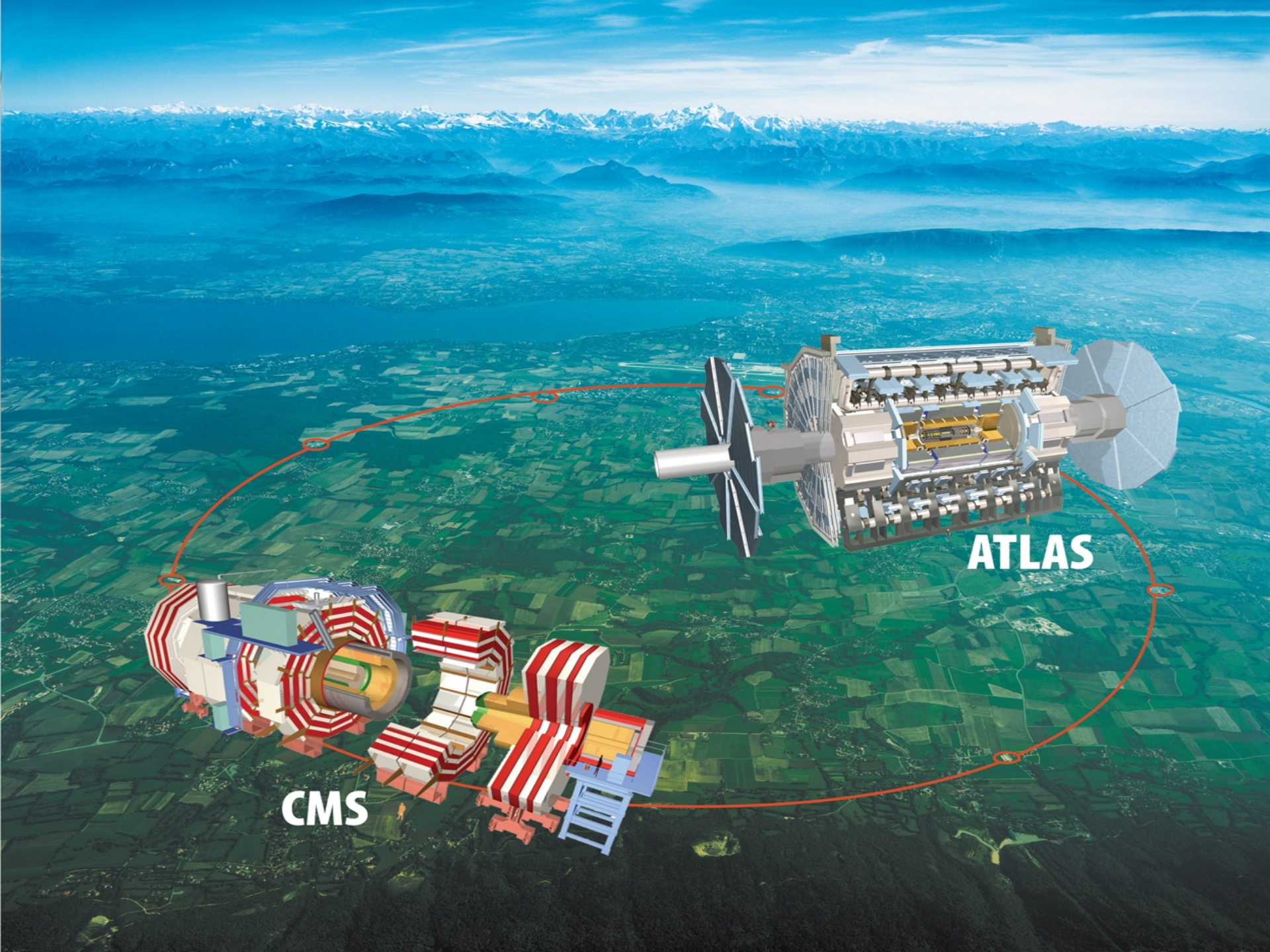


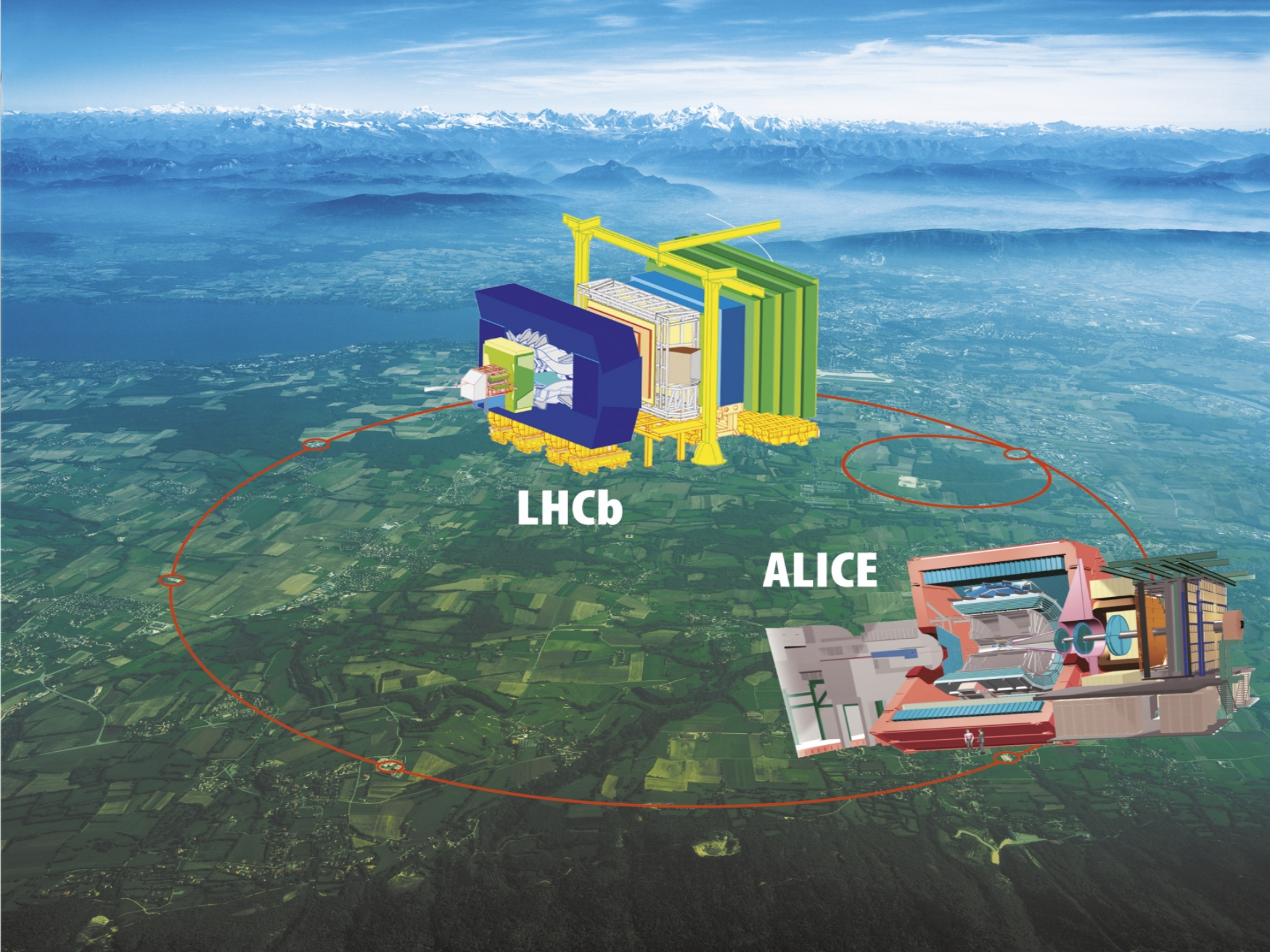

LHCb
ATLAS
CERN Meyrin
CERN Prévessin
SPS 7 km
SUISSE
FRANCE
CMS
LHC 27 km


CMS
ALICE
LHC
LHCb
North Area
SPS
TI8
AWAKE
ATLAS
TT10
TI2
West Area
AD
TT60
TT2
PSB
ISOLDE
East Area
n-TOF
LINAC 2
PS
CTF3
LINAC 3
protons
antiprotons
ions
electrons
neutrons
neutrinos
PS Proton Synchrotron
SPS Super Proton Synchrotron
LHC Large Hadron Collider
AD Antiproton Decelerator
n-TOF Neutron Time Of Flight
AWAKE Advanced Wakefield Experime
CTF3 CLIC Test Facility 3




CERN


ATLAS
CMS


LHCb
ALICE

ROOT - Einleitung

ROOT

- Framework zum Prozessieren grosser Datenmengen
  - Data @ LHC: > 10 PetaByte / Jahr und Experiment
  - Selektion von wenigen Ereignissen aus 10^{12} Kandidaten
- stellt statistische Analyse Algorithmen zur Verfügung
- mathematische Bibliothek mit nicht trivialen und optimierten Funktionen
- Multivariante Analyse Werkzeuge und neuronale Netze
- Werkzeug zur Visualisierung von Daten und Experimentgeometrie
- Interface zur Simulation von physikalischen Ereignissen in Detektoren
- Plattform für das parallele Bearbeiten von Ereignissen (PROOF)
- Implementiert auf unterschiedlichen Betriebssystemen
  Linux, MacOS X, Windows

RootInstall.pdf

ROOT - Start

ROOT setup: Installationspfade von ROOT müssen den environment variablen $PATH und $LD_LIBRARY_PATH hinzugefügt werden

```
export ROOTSYS=/cern/root
export PATH=.:$PATH:$ROOTSYS/bin
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$ROOTSYS/lib
```

Check: `echo $PATH`
`echo $LD_LIBRARY_PATH`

Im CIP Pool bitte ausführen:
`$> source setroot.sh`

ROOT startup im Terminal Fenster mit

```
$> root
root [0] _
Root [1] .? or .help keyword
root [2] .q
root [3] qqqqq
```

öffnet browser

ROOT history:
`$HOME/.root_hist`

- root prompt versteht C++ mit dem Interpreter CINT (V5.xx) / CLing (V6.xx)
  Python binding über PyROOT mit "import ROOT"
  Benutzen Sie die keys ⇧ ⇩ um Befehle erneut zu verwenden
- einfache Benutzung als Taschenrechner auch mit definierten Funktionen
- Darstellen von Funktionen

```
root [4] TF1 *f1 = new TF1("f1","[0]*sin([1]*x)/x",0.,10.);
root [5] f1->SetParameter(0,1); f1->SetParameter(1,1);
root [6] f1->Draw();
```

ROOT – as Calculator

```
marks@jma:~> root
   ------------------------------------------------------------------
  | Welcome to ROOT 6.06/02                        http://root.cern.ch |
  |                                  (c) 1995-2014, The ROOT Team     |
  | Built for linuxx8664gcc                                           |
  | From heads/master@v6-07-02-437-gb06340c, Mar 02 2016, 19:01:57    |
  | Try '.help', '.demo', '.license', '.credits', '.quit'/'.q'        |
   ------------------------------------------------------------------

root [0] gROOT->GetVersion()
(const char *) "6.06/02"
root [1] double x=.5
(double) 0.500000
root [2] int N=30 ; double gs = 0 ;
root [3] for(int i=0;i<N;i++) gs+=TMath::Power(x,i)
root [4] TMath::Abs(gs - (1-TMath::Power(x,N-1))/(1-x))
(Double_t) 1.86265e-09
root [5] double val = 0.992 ;
root [6] sin(val)
(double) 0.837122
root [7] TMath::Pi()
(Double_t) 3.14159
root [8] .!pwd
/local/home/marks
root [9] .!whoami
marks
root [10]
```

Get functions from TMath:
`TMath::function`

Execute system commands:
`.!<unix command>`

ROOT – as Calculator

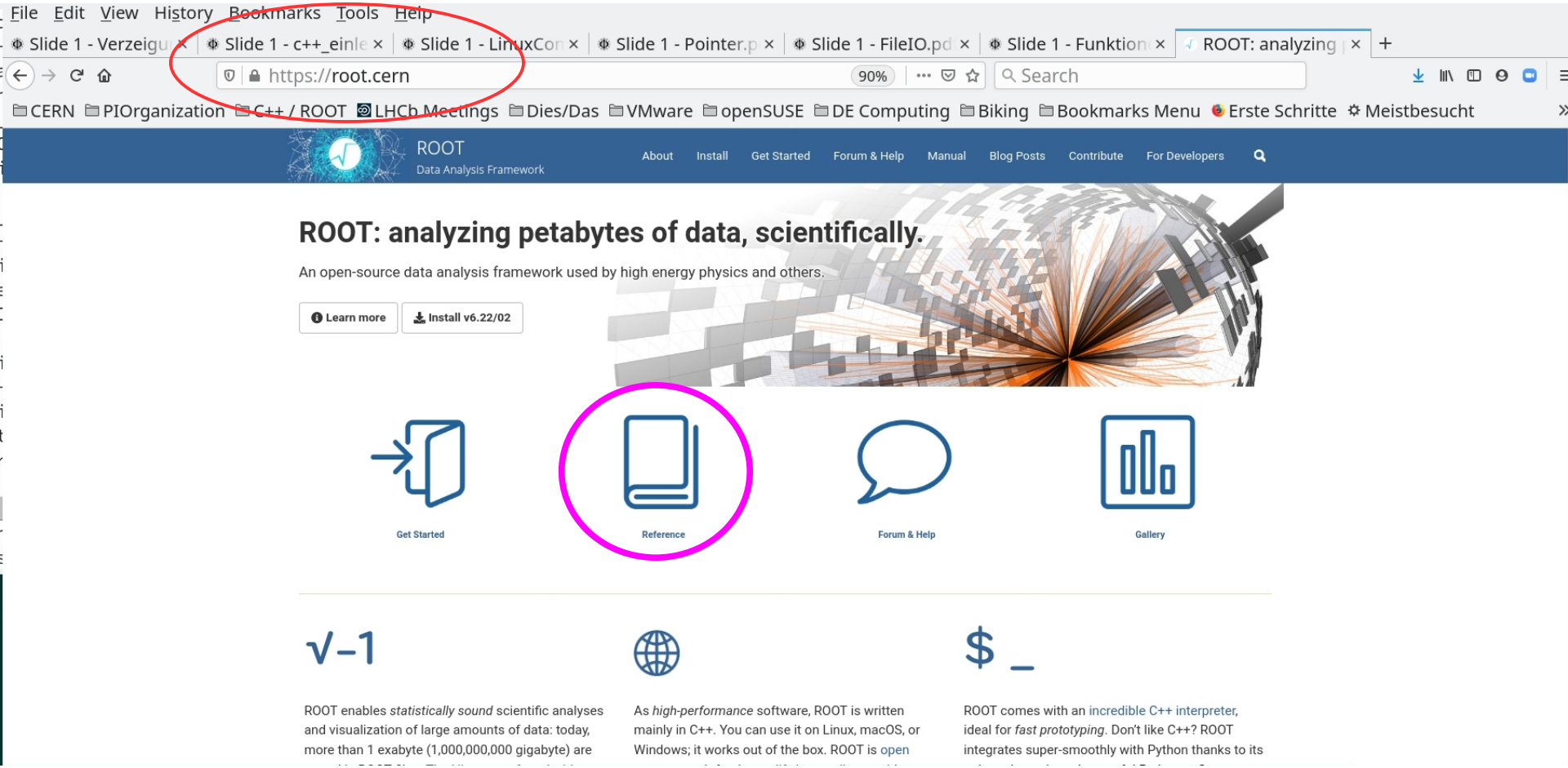
```
marks@jma:~> root
   ------------------------------------------------------------------
  | Welcome to ROOT 6.06/06                        http://root.cern.ch |
  |                                      (c) 1995-2016, The ROOT Team |
  | Built for linuxx8664gcc                                           |
  | From heads/v6-06-00-patches@v6-06-04-66-gb9c1d82, Jul 06 2016, 18:28:55 |
  | Try '.help', '.demo', '.license', '.credits', '.quit'/'.q'        |
   ------------------------------------------------------------------

root [0] TVector3 a(6.,10,40);
root [1] TVector3 b(1.,1,40);
root [2] TVector3 c = a + b ;
root [3] TVector3 d = a.Cross(b);
root [4] double s = a.Dot(b);
root [5] c(0)
(Double_t) 7.00000
root [6] c(1)
(Double_t) 11.0000
root [7] c(2)
(Double_t) 80.0000
root [8] a.Angle(b)
(Double_t) 0.249542
root [9] aperp = a.Orthogonal();
root [10] e = aperp.Unit();
root [11] a.RotateY(TMath::Pi());
root [12] a.Rotate(TMath::Pi()/4, c);
root [13] a(0)
(Double_t) -16.0374
root [14] a(1)
(Double_t) 3.79044
root [15] a(2)
(Double_t) -38.2679
```

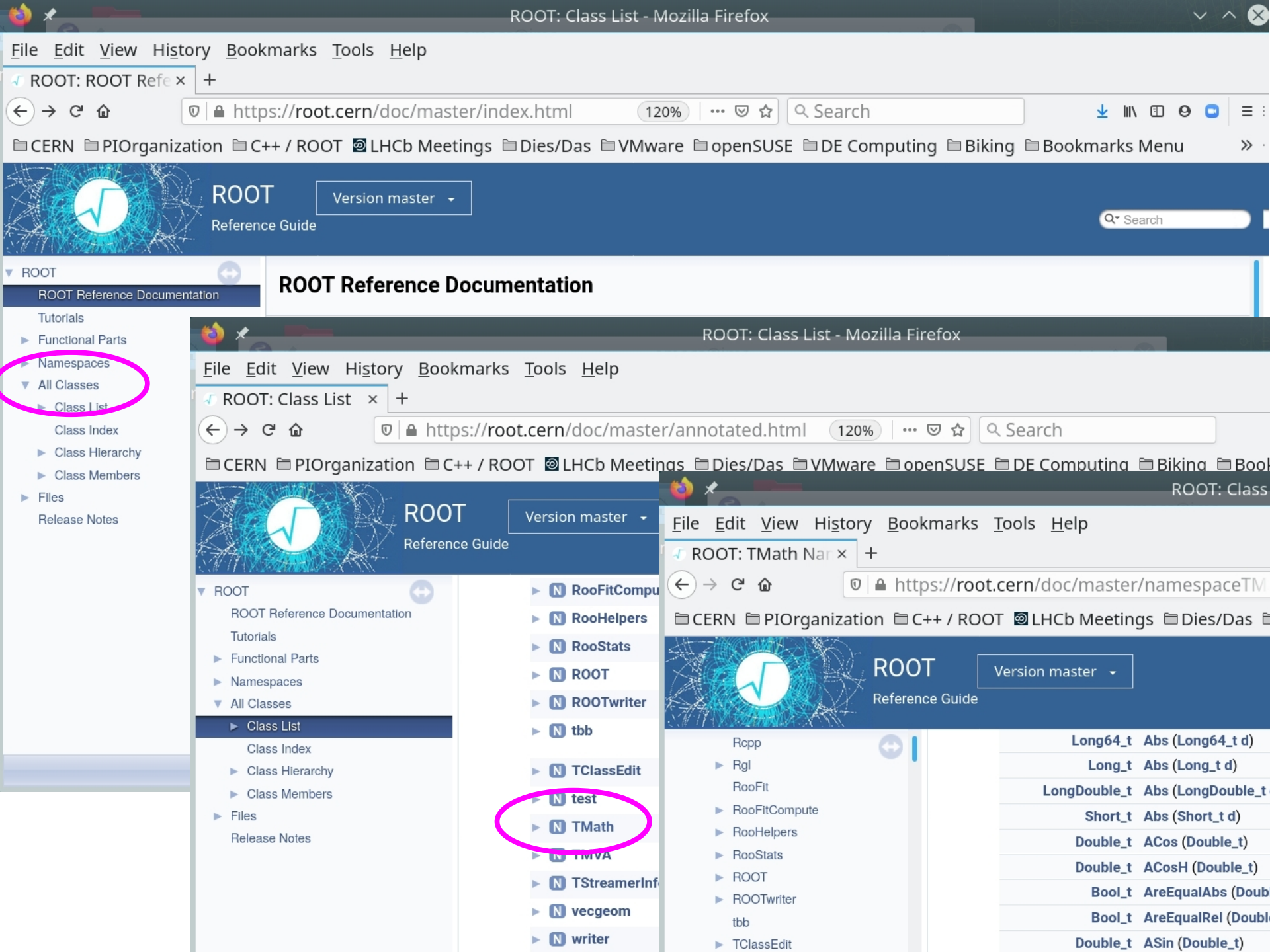
Vektor Rechnung mit dem C++ Interpreter in ROOT

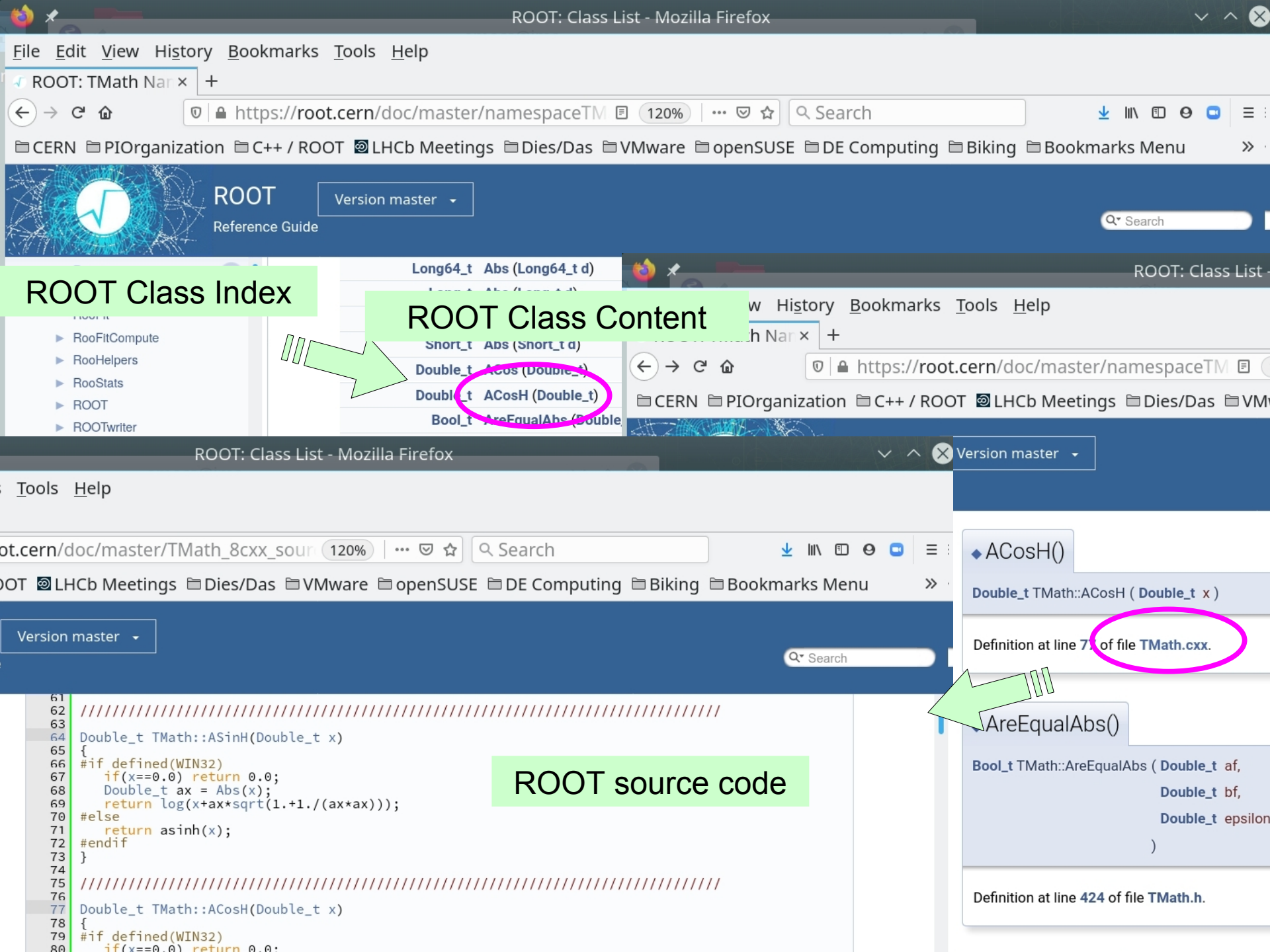
Offensichtlich benutzen wir die Klasse `TVector3`. Woher wissen wir welche Methoden implementiert sind?

```
root[16] .class TVector3
```

ROOT – Dokumentation




ROOT: Class List - Mozilla Firefox
File Edit View History Bookmarks Tools Help
ROOT: ROOT Refe
https://root.cern/doc/master/index.html
120%
Search
CERN PIOrganization C++ / ROOT LHCb Meetings Dies/Das VMware openSUSE DE Computing Biking Bookmarks Menu
ROOT
Version master
Reference Guide
Search
ROOT
ROOT Reference Documentation
Tutorials
Functional Parts
Namespaces
All Classes
Class List
Class Index
Class Hierarchy
Class Members
Files
Release Notes
ROOT Reference Documentation
ROOT: Class List - Mozilla Firefox
File Edit View History Bookmarks Tools Help
ROOT: Class List
https://root.cern/doc/master/annotated.html
120%
Search
CERN PIOrganization C++ / ROOT LHCb Meetings Dies/Das VMware openSUSE DE Computing Biking
ROOT
Version master
Reference Guide
ROOT
ROOT Reference Documentation
Tutorials
Functional Parts
Namespaces
All Classes
Class List
Class Index
Class Hierarchy
Class Members
Files
Release Notes
RooFitCompu
RooHelpers
RooStats
ROOT
ROOTwriter
tbb
TClassEdit
test
TMath
TMVA
TStreamerInf
vecgeom
writer
ROOT: Class
File Edit View History Bookmarks Tools Help
ROOT: TMath Nar
https://root.cern/doc/master/namespaceTM
CERN PIOrganization C++ / ROOT LHCb Meetings Dies/Das
ROOT
Version master
Reference Guide
Rcpp
Rgl
RooFit
RooFitCompute
RooHelpers
RooStats
ROOT
ROOTwriter
tbb
TClassEdit
Long64_t Abs (Long64_t d)
Long_t Abs (Long_t d)
LongDouble_t Abs (LongDouble_t
Short_t Abs (Short_t d)
Double_t ACos (Double_t)
Double_t ACosH (Double_t)
Bool_t AreEqualAbs (Doub
Bool_t AreEqualRel (Doubl
Double_t ASin (Double_t)




```
61
62  //////////////////////////////////////////////////////////////////////////////
63
64  Double_t TMath::ASinH(Double_t x)
65  {
66  #if defined(WIN32)
67     if(x==0.0) return 0.0;
68     Double_t ax = Abs(x);
69     return log(x+ax*sqrt(1.+1./(ax*ax)));
70  #else
71     return asinh(x);
72  #endif
73  }
74
75  //////////////////////////////////////////////////////////////////////////////
76
77  Double_t TMath::ACosH(Double_t x)
78  {
79  #if defined(WIN32)
80     if(x==0.0) return 0.0;
```

ROOT - Start

Graphische Darstellung von Daten aus einem File

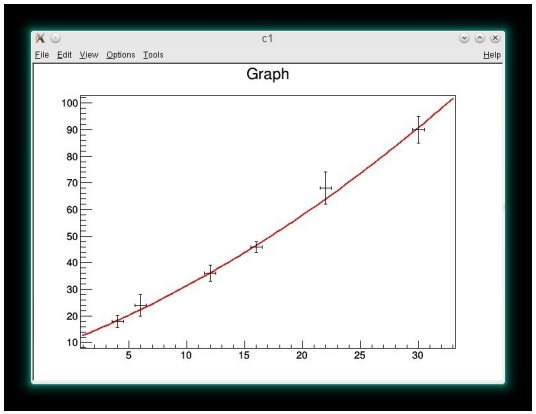
```
$> root
root [0] TGraphErrors *gr = new TGraphErrors("myFile.txt");
root [1] gr->SetTitle("myTitle;xAxis;yAxis");
root [2] gr->Draw("AP");
```

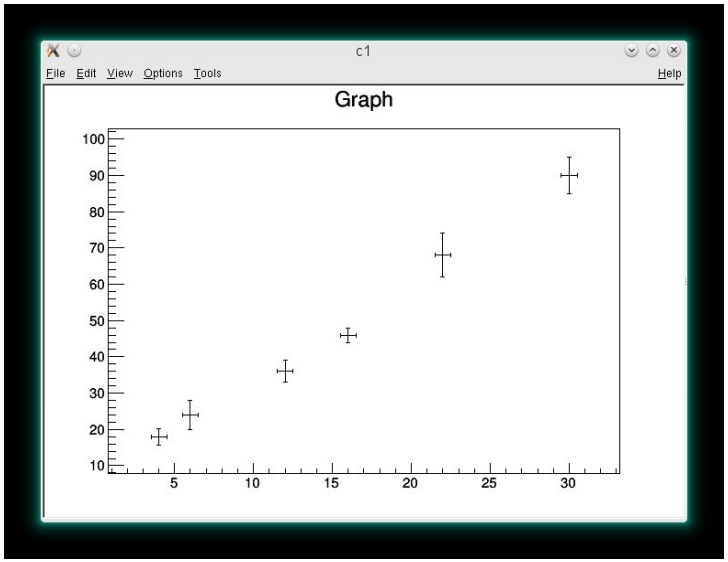
Draw **A**xis and **P**oints

myFile.txt :

```
4    18   0.5  2.3
6    24   0.5  4.1
12   36   0.5  3.0
16   46   0.5  2.0
22   68   0.5  6.0
30   90   0.5  5.0
```

- Graphik kann modifiziert werden:
  - Symbole
  - Achsenbeschriftung / Tick marks
  - beliebiger Text
- Als root file speichern zur weiteren Bearbeitung
- Parameter anpassen mit dem
  Fit Panel unter dem Menue „Tools“





ROOT – Wo arbeiten wir?

- Im CIP Pool → Installation aktuell allerdings 2 Jahre alt
  - Start Cisco AnyConnect
  - Start MobaXterm oder Linux shell
    In MobaXterm shell: `ssh myUserid@physik1.kip.uni-heidelberg.de`
- Joergs Desktop Rechner → neueste ROOT Version, für das python Interface notwendig.
  - Start MobaXterm oder Linux shell
  - Login mit CIP Pool userids mit dem password !KIP227
    CIP home dir nicht vorhanden → transfer data mit scp wenn notwendig
    `ssh myUserid@zenon.physi.uni-heidelberg.de`
  - Beim ersten login:
    I) testen ob das keyboard geht → ist y wirklich y
    II) root testen → `root` in der shell eintippen
    III) password ändern → `passwd` eintippen und neues password setzen
- Eigene ROOT Installation RootInstall.pdf

ROOT – Wo arbeiten wir?

- Auf dem host zenon.physi.uni-heidelberg.de
  - 500 GB für alle im home dir
  - alle üblichen Linux Programme installiert
  - python3 benutzen
  - c++ compiler g++ , NVidia CUDA code über nvcc
  - Jupyter notebooks mit automatisch öffnendem browser

```
zenon:~> jupyter-notebook
```

  - ROOT mit jupyter notebook support

```
zenon:~> root --notebook
```

  - chrome Browser mit google-chrome
  - editor: emacs, kate, vi
  - X Verbindungen langsam → Geduld
  - Login zum CIP pool mit physik1, physik2, ….
  - Datentransfer mit scp:

vom local File `myFile.txt` auf physik1 ins dir `myDir`

```
scp myFile.txt myUserid@physik1.kip.uni-heidelberg.de:myDir/.
```

Wir wollen den ROOT Interpreter und die ROOT Klassen verwenden, um das Rechnen mit Vektoren zu demonstrieren und ausserdem Funktionen darzustellen.

Arbeitsvorschlag:

- Benutzen Sie den ROOT Interpreter mit einfachen C++ keywords. So könnten Sie zum Beispiel die Summe $\sum_{k=1}^{N} \frac{x^k}{k}$ für N=10 und x = 3 bestimmen.

  Was ist umständlich?

- Benutzen Sie die Klasse `TLorentzVector` um die invariante Masse der Vierer-vektoren `u(3.,10.,40.,10.)` und `v(1.,0.,10.,3.)` zu bestimmen. Bestimmen Sie die transversale Komponente des Summenvektors von u und v.

- Laden Sie das Text File myFile.txt mit gedachten Messungen und benutzen Sie die Klasse `TGraphErrors("myFile.txt")` zur Darstellung. Probieren Sie die Fit Funktionalität aus.

- In Ihrem home Verzeichnis gibt es ein File `.root_hist`. Was befindet sich in dem File?

ROOT - Start

Ausführung von Programmen

```
$> root
root [0] .x myMacro.C(2,2)
```

- File myMacro.C wird von CLing interpretiert und ausgeführt

```
$> root
root [0] .x myMacro.C+(2,2)
root [1] myMacro(2,2)
```

myMacro.C:

```
int myMacro(int a, int b)
{
   int s = a + b ;
   return s ;
}
```

- File myMacro.C wird mit ACLiC kompiliert und es wird eine shared library erzeugt myMacro.so
  - system compiler wird benutzt
  - das File wird nur kompiliert, wenn es geändert wurde.
- CINT/CLing versus compiled C++
  - Interpreter → rapid prototyping
  - Compiled code → Syntax check, schneller bei der Ausführung

ROOT - Start

Ausführung von Programmen

- Gemeinsames Ausführen von ROOT und C++ code

```
$> g++ -o throwDice throwDice.cc  `root-config --cflags --glibs`
$> ./throwDice 3
Dice 1: 3
Dice 2: 4
Dice 3: 4
```

Programm wird in der shell unter Hinzufügen der ROOT spezifischen flags kompiliert

`command`: command wird ausgefuehrt

-o FileName : FileName des Programms

```
$> root
root [0] .x throwDice.cc(3)
Dice 1: 5
Dice 2: 4
Dice 3: 2
```

Programm wird in ROOT von CLing interpretiert

```
root [1] .x throwDice.cc+(3)
root [2] throwDice(3)
Dice 1: 2
Dice 2: 2
Dice 3: 2
```

Programm wird in ROOT mit ACLiC kompiliert und eine shared library gebaut, es kann dann in ROOT ausgeführt werden

File Edit Options Buffers Tools C++ Help

Save Undo

```
# include <cstdlib>
# include <iostream>
# include "TRandom3.h"

using namespace std;

# ifndef __CINT__  // the following code will be invisible for CINT interpreter
int rollDice();
void throwDice (int NDice);

int main(int argc, char* argv[])
{
  throwDice (atoi(argv[1])) ;
  return 0 ;
}
# endif   // end ignore

void throwDice (int NDice) {
  for (int I = 1 ; I <= NDice; I++)
     cout << "Dice " << I << ": " << rollDice() << endl;
  return ;
}
int rollDice() {
   UInt_t MaxInt = 6 ;
   TRandom3 *R = new TRandom3();
   R->SetSeed(0);  // set seed to machine clock
   UInt_t NRnd  = R->Integer(MaxInt) ;
   int roll = static_cast <int> (NRnd) + 1 ;
   return roll ;
}
```

-:--- **throwDice.cc** Top (4,0) (C++/l Abbrev)

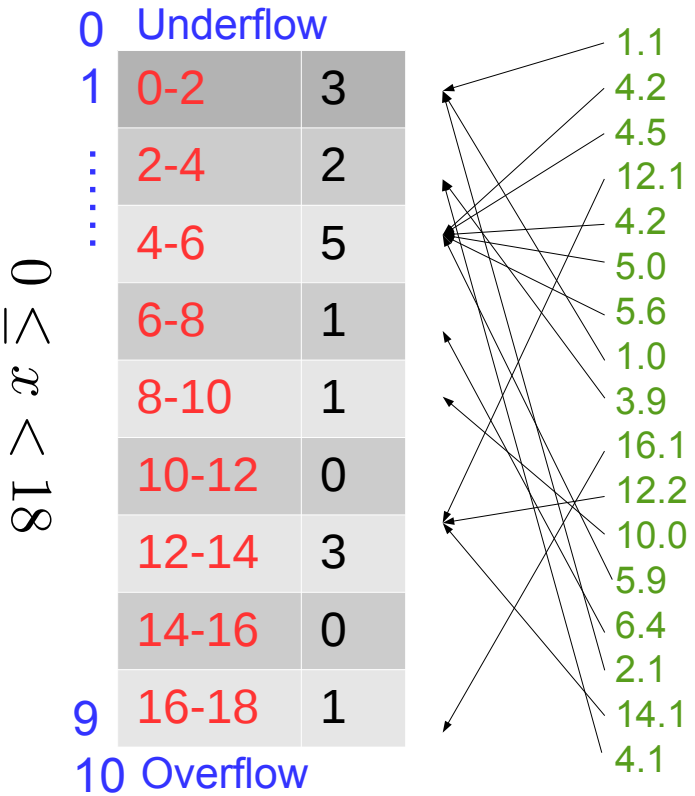
throwDice.cc

https://root.cern/doc/master/classTRandom3.html

Arbeitsvorschlag: Können Sie das Programm so ändern, dass es bevorzugt 3 würfelt.

ROOT - Histogramme

Histogramm: In Abschnitte geteiltes Diagramm bezüglich einer Variablen, in das die Anzahl der Werte für jeden Abschnitt eingetragen wird



- Histogramm Klassen in ROOT:
  TH1F, TH1D (1 dimensional)
  TH2F, TH2D (2 dimemsional)
- Benutzung:
  - Initialisieren
  - Variable füllen
  - Graphische Darstellung und Bearbeitung
  - Resultate in einem ROOT File speichern

Mean

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Standard Deviation

$$s = \sqrt{\frac{1}{n-1}\left[\left(\sum_{i=1}^{n} x_i^2\right) - \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2\right]}$$

Skewness

$$S = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_i - \bar{x}}{s}\right)^3$$

Curtosis

$$C = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_i - \bar{x}}{s}\right)^4$$

Mean +/- Error
Standard Deviation
Skewness
Curtosis

ROOT - Histogramme

Histogramm: In Abschnitte geteiltes Diagramm bezüglich einer Variablen, in das die Anzahl der Werte für jeden Abschnitt eingetragen wird

- Beispiel Macro der Histogramm Klassen in ROOT

myHisto.C

  - Initialisieren

```
TH1F *h = new TH1F ("myhist","Altersverteilung",60,0.,60.);
```

pointer auf das TH1F Objekt h | Histogramm Name | Titel | Anzahl der Bins | Grenzen

  - Variable füllen

```
h->Fill(age,weight=1.);
```

  - Graphische Darstellung und Bearbeitung

```
h->Draw("Options");
Options = "","E","SAME","C","L"
```

Viele Optionen und Beispiele in der Klassenreferenz:
https://root.cern/doc/master/classTHistPainter.html#HP01

Fehlerbalken | in den gleichen Plot | glatte Kurve | Line

  - Resultate in einem ROOT File speichern

```
TFile outFile ("myFile.root","RECREATE");
```
Outputfile öffnen

```
h→Write();
```
Histogramm schreiben

```
outFile.Close();
```
Outputfile schließen

ROOT - Histogramme

Histogramm: In Abschnitte geteiltes Diagramm bezüglich einer Variablen, in das die Anzahl der Werte für jeden Abschnitt eingetragen wird

- Ausführen des Beispiel Macros und Ansehen des Histogramms

myHisto.C

```
$> root myHisto.C
root [0]
Processing myHisto.C...
Histogram filled with 20 entries
root [0] .q
$> ls altersverteilung.root
altersverteilung.root
$> root  --web=off altersverteilung.root
Root [0] TBrowser T
```

Programm wird in ROOT von CLing interpretiert

Enthält ein ROOT file mit Histogramm

Mit dem Browser läßt sich das ROOT file ansehen und das Histogramm darstellen. Mit --web=off wird der ROOT mode verwendet

File Edit Options Buffers Tools C++ Help

Save Undo

```
# include <iostream>
# include <TRandom3.h>
# include <TFile.h>
# include <TH1.h>

using namespace std;

void myHisto() {
   int NumberParticipants = 20 ;
   Double_t averageAge= 20. , sigmaAge = 1.0 ;
   Double_t age;

   TRandom3 *R = new TRandom3();
   R->SetSeed(0);  // set seed to machine clock

   TH1F *h = new TH1F ("myhist","Altersverteilung",,0.,60.);

   for (int I = 0 ; I < NumberParticipants ; I++) {
       age = R->Gaus(averageAge,sigmaAge) ;
       h->Fill(age);
   }

   cout << "Histogram filled with "<< NumberParticipants<< " entries" << endl;

   TFile outFile ("altersverteilung.root","RECREATE");
   h->Write();
   outFile.Close();

   return ;
}
```

myHisto.C

-:--- **myHisto.C** Top (5,0) (C++/l Abbrev)

ROOT - Histogramme

- Graphische Darstellung im Programm

  Die graphische Ausgabe von ROOT wird in einem “Canvas“ dargestellt, das vom Displaymanager kontrolliert wird. Es kann in mehrere Canvas Fenster geschrieben werden. Ein Canvas kann in Unterbereiche geteilt werden.

```
TCanvas *C = new TCanvas("myC","Pad Histogram",0,0,800,600);
```


```
C->Divide(2,2);
```

Erzeugen eines Pads mit 2x2 Subfeldern zum Darstellen von Plots

```
C->cd(1);
h->DrawClone();
```

Darstellen des Histogramms h und erzeugen einer Kopie

```
C->cd(2);
Double_t NumEntries = h->Integral();
h->Scale(1./NumEntries);
h->Draw();
```

Normieren der Anzahl der Histogramm-einträge von h auf 1

```
C->Write();
```

Schreiben des Canvas in das ROOT file

ROOT - Histogramme

- Histogramm Ergebnisse im Programm weiter verwenden und setzen

```
Double_t binContent = h->GetBinContent(22);
Double_t error = h->GetBinError(22);
```

Get Inhalt Bin 22

```
Double_t NumEntries = h->GetEntries();
Int_t MaxBin = h->GetMaximumBin();
Double_t histoMean = h->GetMean();
Double_t histoMeanError = h->GetMeanError();

TAxis *xaxis = h->GetXaxis();
Double_t binCenter = xaxis->GetBinCenter(22);

h->GetXaxis()->SetTitle("X axis title");
h->GetYaxis()->SetTitle("Y axis title");

h->SetTitle("Histogram title;  Another X title Axis");
```

- Histogramm speichern und lesen

```
TFile f("histos.root", "new")
h->Write();
```

Schreiben des Histograms in das ROOT file

```
TFile f("histos.root");
TH1F *h = (TH1F*)f.Get("hgaus");
```

Lesen des Histograms

ROOT - Histogramme

- Operationen mit Histogramm

myHistoExtended.C

```
TH1F *h = new TH1F ("myhist","Altersverteilung",60,0.,60.);
h->GetXaxis()->SetTitle("age[years]");
h->Sumw2(); // get access to proper error treatment

TH1F *hbck = new TH1F ("hbck","Untergrund
                                Altersverteilung",60,0.,60.);
hbck->GetXaxis()->SetTitle("age[years]");
hbck->Sumw2();

TH1F *hsum  =(TH1F*)h->Clone();
hsum->SetName("hsum");
hsum->Add(hbck,1.0);

TH1F *h3 = new TH1F ("h3","h1+h2",nBins,xMin,xMax);
h3->Add(h1,h2);
h3->Add(h1,h2,1.,-1.);
```

Erzeugen einer Kopie des Histogramms h

Addieren von hbck zu h mit Skalenfaktor 1

Addieren von 2 Histogrammen

Subtrahieren von 2 Histogrammen

- Histogramme h1,h2,h3 haben das gleiche binning!
- Sonst müssen Bingrenzen erhalten bleiben

ROOT - Histogramme

myHistoExtended.C

- Histogramme in 2D

```
TH2F *h2D = new TH2F("h2D", "Alter vs Groesse",
                                          60,0.,60.0,50,140,190);
h2D->GetXaxis()->SetTitle("age[years]");
h2D->GetYaxis()->SetTitle("height[cm]");
h2D->Sumw2();
```

Optionen zur Darstellung

```
TH2F *h = new TH2F("h","2D Gauss",40,-4.,4.,40,-4.,4.);
for(int j=0; j<10000;j++){
   double x = R->Gauss(0,1);
   double y = R->Gauss(1,2);
   h->Fill(x,y);
}
```

Gauss Verteilung, Mean=0 und Breite =1

Gauss Verteilung, Mean=1 und Breite =2

```
h->Draw("COLZ");
h->Draw("CONTZ");
h->Draw("LEGO");
```

Farbänderungen entsprechend der Einträge in z

Contour Plot bezüglich z

Lego Plot Entries in z

https://root.cern/doc/master/classTHistPainter.html#HP01

ROOT - Histogramme

- Histogramme in 2D

```
TH2F *h2D = new TH2F("h2D", "Alter vs Groesse",
                                    60,0.,60.0,50,140,190);
```

Projektionen auf die Achsen erzeugen 1D Histogramme

```
TH1F *projX =  h2D->ProjectionX("MyProjX",1,-1);
TH1F *projY =  h2D->ProjectionY("MyProjY",1,nBin);
```

Binbereich: -1 =: kopieren bis zum Ende

Profil von 2D Histogrammen: Mittelwerte in Bins bezüglich einer Koordinate

```
TProfile *profX =  h2D->ProfileX("MyProfileX",1,
TProfile *profY =  h2D->ProfileY("MyProfileY",1,
```

Binbereich in y/x

Hier wird die Bin Grösse des 2D Histogramms verwendet. Für die Mittelwertbildung werden nur die Bin Mitten benutzt! Daher haben wir nur eine Näherung.

TProfile: bilde Mittelwerte von y in Bins von x

```
TProfile *prof = new TProfile("prof","Title",
                                    100,0.,60.0,140,190);

prof→Fill(x,y);
```

Bins von x. x Bereich y Bereich

Mittelwert: y Fehler: $rms/\sqrt{n}$

Datenanalyse – erste Schritte

Wir wollen nun beispielhaft eine Messung analysieren, bei der das Ausgangssignal eines Verstärkers wiederholt gemessen wird. Der Messbereich ist [0,450]. Gleichzeitig wird mit jeder Messung die Raumtemperatur aufgezeichnet. Es werden 2 Textfiles geschrieben, signal.txt und temperature.txt

Arbeitsvorschlag:

- Schreiben Sie ein ROOT Macro, das das Verstärkersignal darstellt. Bestimmen Sie die Anzahl der Messungen, den Mittelwert und den Fehler und die Standard-abweichung. Ist das Signal gaussverteilt? Welcher Bereich ist als Signalbereich sinnvoll? Mit welcher Auflösung wird das Signal gemessen?
- Schätzen Sie den Anteil der Untergrundeinträge im Signalbereich ab?
- Gibt es eine Korrelation zwischen Signal und Temperatur. Was bedeutet das für die Auflösung der Signalverteilung. Wie kann man die Temperaturabhängigkeit des Signals korrigieren? Wie groß ist dann die Auflösung?
- Subtrahieren Sie die Untergrundverteilung von der korrigierten Signalverteilung. Wie groß ist nun die Auflösung?

analysis.cc

Datenanalyse – erste Schritte

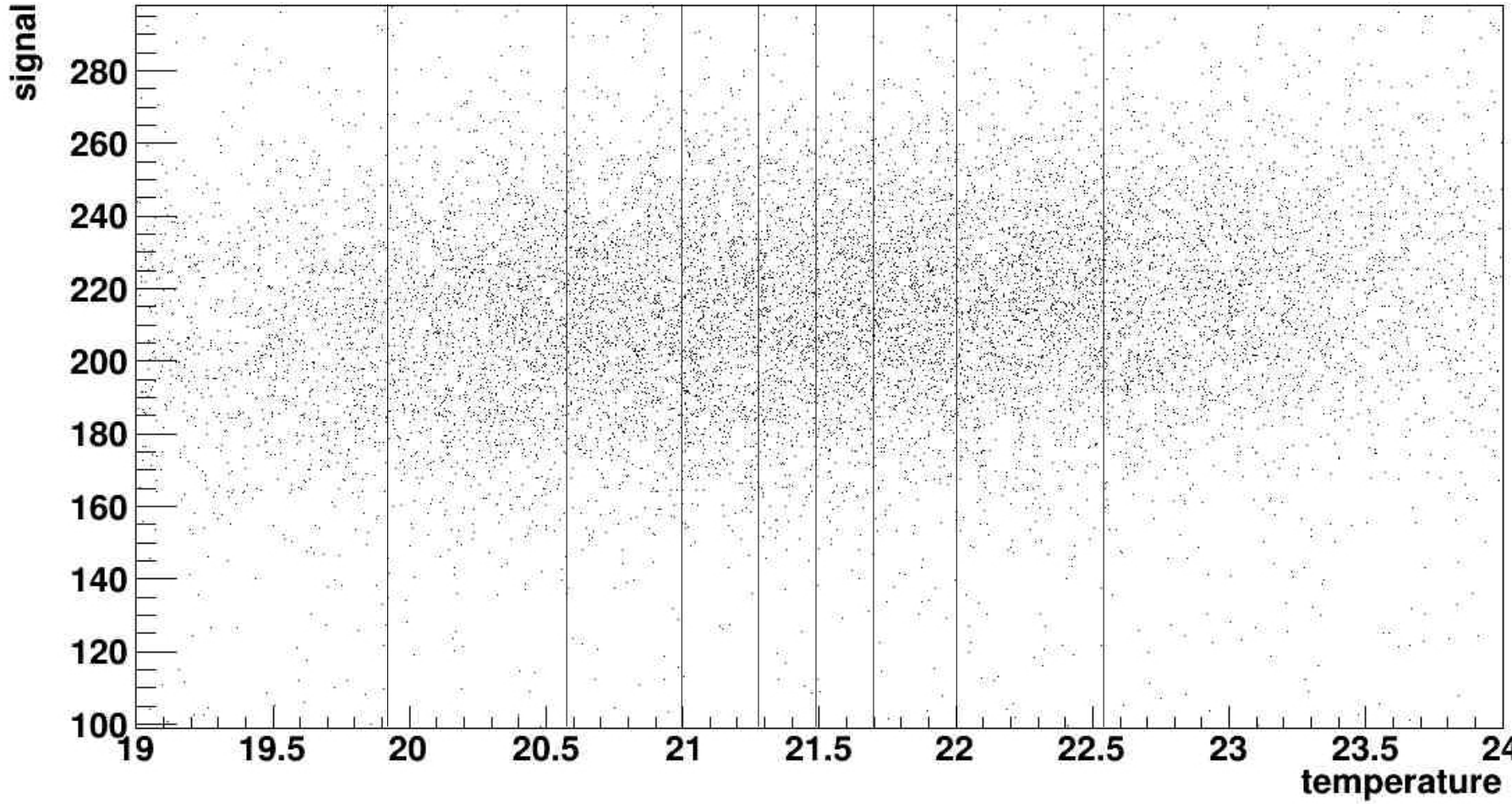
Wir wollen nun beispielhaft eine Messung analysieren, bei der das Ausgangssignal eines Verstärkers wiederholt gemessen wird. Der Messbereich ist [0,450]. Gleichzeitig wird mit jeder Messung die Raumtemperatur aufgezeichnet. Es werden 2 Textfiles geschrieben, signal.txt und temperature.txt

Lösungsschritte:

- Öffnen der Textfiles mit 2 Eingabeströmen
- Lesen jeweils bis EOF und speichern als vector <double>
- 2 1D Histogramme mit Signal und Temperatur, 1 2D Histogramm Temperatur gegen Signal auftragen.
- Interaktiv Gauss Anpassung, Auflösung = Sigma / Mean
- Integriere Signal im Bereich [350 – 450] (nur Untergrund). Da der Untergrund gleichverteilt ist, kann damit auf den Untergrund im Signalbereich geschlossen werden.
- Im 2D Plot ist klar eine Korrelation zwischen Signal und Temperatur sichtbar. Die Auflösung ist dadurch überschätzt.
- Bestimme das mittlere Signal in 4 Bins der Temperatur (interaktive Gauss Anpassung). Daraus kann die Änderung mit der Temperatur bestimmt werden.
- Diese Änderung kann durch die Umrechnung des Signals auf T=19 Grad korrigiert werden. Die korrigierten Werten werden als vector gespeichert und histogrammiert.
- Damit kann die Auflösung bestimmt werden.
- Wir kennen die Zahl der Untergrundereignisse im gesamten Signal Bereich. Erzeuge ein Untergrund Histogramm in das Zufallswerte gefüllt werden. Das Untergrund Histogramm kann man abziehen.

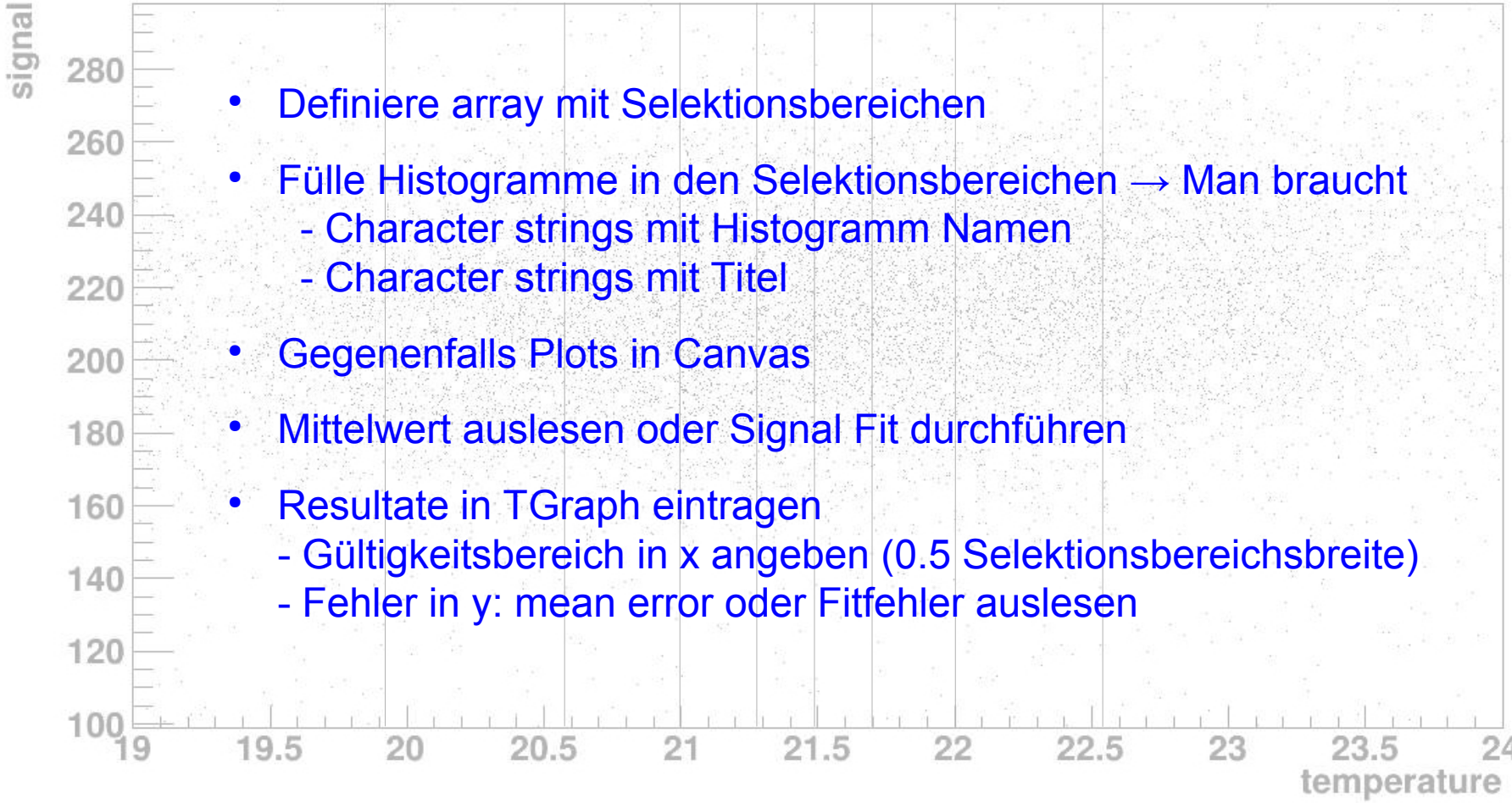
ROOT - Histogramme

Im Analyse Programm müssen oft viele Histogramme einer Messgrösse erzeugt werden und Mittelwerte / Fehler oder Fit-Werte / Fehler in beliebigen Bereichen von anderen Variablen ermittelt und weiter verwendet werden. Im folgenden werden einige häufig verwendbare Anweisungsblöcke gezeigt.



ROOT - Histogramme

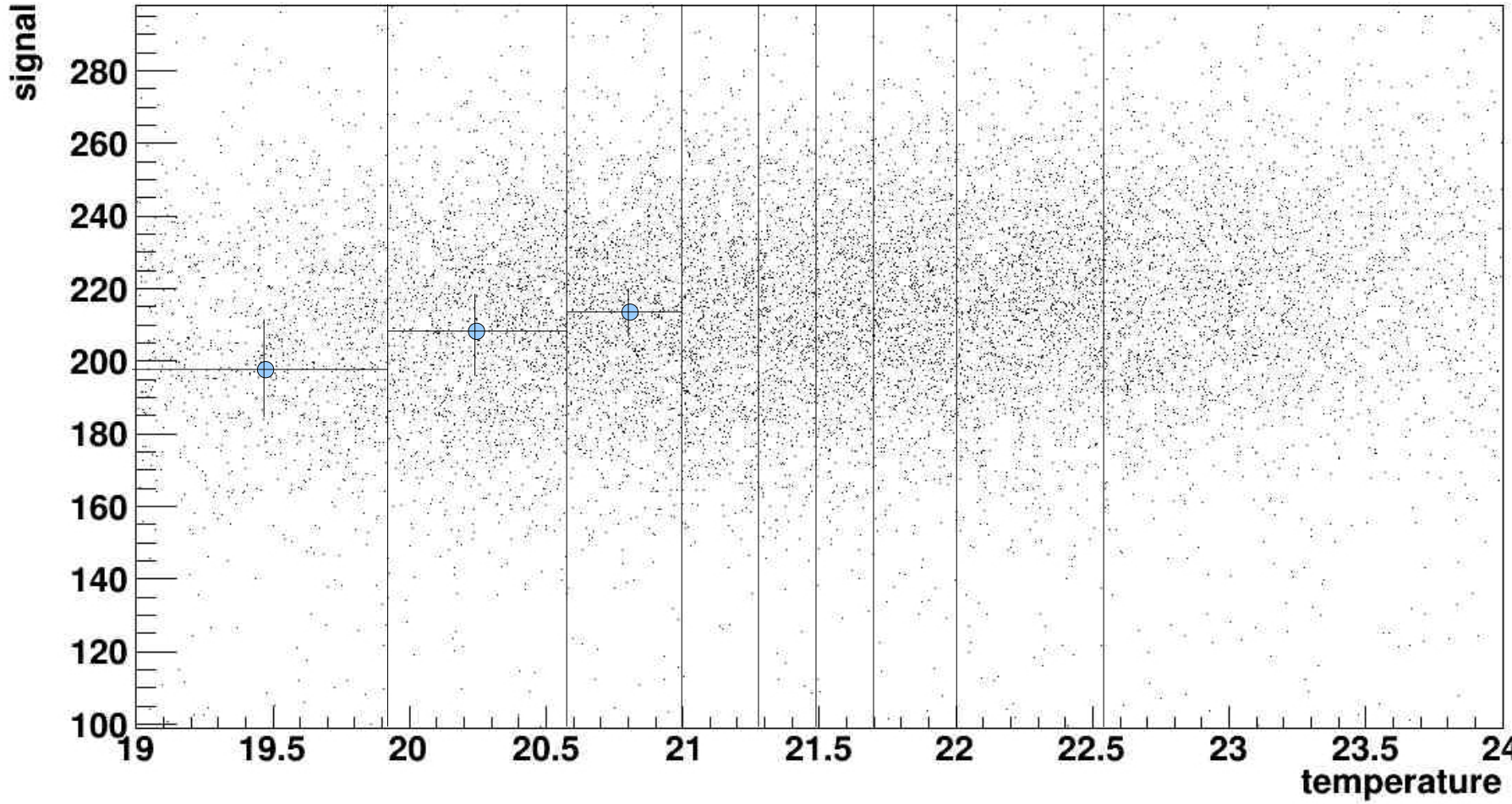
Im Analyse Programm müssen oft viele Histogramme einer Messgrösse erzeugt werden und Mittelwerte / Fehler oder Fit-Werte / Fehler in beliebigen Bereichen von anderen Variablen ermittelt und weiter verwendet werden. Im folgenden werden einige häufig verwendbare Anweisungsblöcke gezeigt.



- Definiere array mit Selektionsbereichen
- Fülle Histogramme in den Selektionsbereichen → Man braucht
  - Character strings mit Histogramm Namen
  - Character strings mit Titel
- Gegenenfalls Plots in Canvas
- Mittelwert auslesen oder Signal Fit durchführen
- Resultate in TGraph eintragen
  - Gültigkeitsbereich in x angeben (0.5 Selektionsbereichsbreite)
  - Fehler in y: mean error oder Fitfehler auslesen

ROOT - Histogramme

Im Analyse Programm müssen oft viele Histogramme einer Messgrösse erzeugt werden und Mittelwerte / Fehler oder Fit-Werte / Fehler in beliebigen Bereichen von anderen Variablen ermittelt und weiter verwendet werden. Im folgenden werden einige häufig verwendbare Anweisungsblöcke gezeigt.



ROOT - Histogramme

Im Analyse Programm müssen oft viele Histogramme einer Messgrösse erzeugt werden und Mittelwerte / Fehler in Bereichen von anderen Variablen ermittelt und weiter verwendet werden. Im folgenden werden einige häufig verwendbare Anweisungsblöcke gezeigt.

Variable Anzahl von Histogrammen einer Messgrösse in definierbaren Bereichen einer anderen Grösse

```
char name[128],title[128];
const int nHisto = 6 ;
double Range[nHisto+1] ={0.,1.5,2.9,3.5,5.0,7.0,9.5};
TH1F *hisSig[nHisto];
for(int j=0; j<nHisto;j++){
sprintf(name,"range%i_%i",(int)Range[j],(int)Range[j+1]);
sprintf(title,"Range: %5.1f<Range<%5.1f",Range[j],Range[j+1]);
hisSig[j] = new TH1F(name,title,nBin,Range[j],Range[j+1]); }

for (int I = 0; I < nEvent; I++){
    for (int j = 0; j < nHisto ; j++) {
      if(S[I] >= Range[j] && S[I] < Range[j+1])
                                    hisSig[j]->Fill(S[I]);}
}
```

definiere Selektionsbereich

array mit Histogramm Zeigern

dynamisches Erzeugen der Histogramme

Füllen der Histogramme in Selektionsbereichen

ROOT - Histogramme

Im Analyse Programm müssen oft viele Histogramme einer Messgrösse erzeugt werden und Mittelwerte / Fehler in Bereichen von anderen Variablen ermittelt und weiter verwendet werden. Im folgenden werden einige häufig verwendbare Anweisungsblöcke gezeigt.

Variable Anzahl von Histogrammen einer Messgrösse in definierbaren Bereichen einer anderen Grösse

```
char name[128],title[128];
const int nHisto = 6 ;
double Range[nHisto+1] ={0.,1.5,2.9,3.5,5.0,7.0,9.5};
TH1F *hisSig[nHisto];
for(int j=0; j<nHisto;j++){
sprintf(name,"range%i_%i",(int)Range[j],(int)Range[j+1]);
sprintf(title,"Range: %5.1f<Range<%5.1f",Range[j],Range[j+1]);
hisSig[j] = new TH1F(name,title,nBin,Range[j],Range[j+1]); }

for (int I = 0; I < nEvent; I++){
    for (int j = 0; j < nHisto ; j++) {
      if(S[I] >= Range[j] && S[I] < Range[j+1])
                                    hisSig[j]->Fill(S[I]);}
}
```

definiere Selektionsbereich

array mit Histogramm Zeigern

dynamisches Erzeugen der Histogramme

Füllen der Histogramme in Selektionsbereichen

ROOT - Histogramme

Bestimmung der Mittelwerte/Fehler und rms von Histogrammen

Fortsetzung der obigen Anweisungen

```
double S_mean[nHisto], S_meanError[nHisto], S_rms[nHisto];
for(int j=0; j<nHisto;j++){
  S_mean[j] = hisSig[j]→GetMean();
  S_meanError[j] = hisSig[j]→GetMeanError();
  S_rms[j] =  hisSig[j]→GetStdDev();
}
```

Methoden um die Histogramm-eigenschaften auszulesen

Wenn die Messungen gaussverteilt sind, lassen sich Mittelwert und Fehler auch durch die Anpassung eines Gauss Models an die Histogramme bestimmen.

```
double Fit_mean[nHisto],Fit_meanErr[nHisto],Fit_sigma[nHisto];
for(int j=0; j<nHisto;j++){

 hisSig[j]→Fit("gaus","0","",Range[j],Range[j+1]);

 Fit_mean[j]=hisSig[j]→GetFunction("gaus")->GetParameter(1);
 Fit_sigma[j]=hisSig[j]→GetFunction("gaus")->GetParameter(2);
 Fit_meanErr[j]=hisSig[j]→GetFunction("gaus")->GetParError(1);}
```

Anpassung des Gauss Models an die Histogramme

(Details im folgenden)

ROOT – Graphische Darstellung

Drei Klassen (TGraph, TGraphErrors, TGraphAsymmErrors) können in ROOT zur graphischen Darstellung von Messungen verwendet werden. Grundsätzlich werden die Zahl der Messpunkte und arrays mit den darzustellenden Werten gegeben.

```
const int nHisto = 6 ;
double Range[nHisto+1] ={0.,1.5,2.9,3.5,5.0,7.0,9.5};
// Werte der Gauss Anpassung
double Mean[nHisto], MeanErr[nHisto];
double RangeBin[nHisto], RangeBinErr[nHisto];
for(int j=0; j<nHisto;j++){
  RangeBin[j] = Range[j]+(Range[j+1]-Range[j])/2. ;
  RangeBinErr[j] = (Range[j+1]-Range[j])/2. ;
}
TGraphErrors *grRangeDep = new
       TgraphErrors(nHisto,RangeBin,Mean,RangeBinErr,MeanErr);
// Draw onto Canvas
  TH1F * frame = gPad->DrawFrame(xMin,yMin,xMax,yMax);
  frame->SetYTitle("Signal from fit[myUnit]");
  frame->Draw("");
  grRangeDep→SetMarkerColor(2);
  grRangeDep→SetMarkerStyle(21);
  grRangeDep→Draw("P");
```

Zeichnen des festen Rahmens

Zeichnen des Graphen

ROOT – Darstellung Histogramm

Mit den folgenden Methoden läßt sich die Histogramm Darstellung modifizieren.

`h→SetLineColor(kRed);`	Linienfarbe auf rot setzen
`h→SetTitle(“my Title“);`	Titel setzen
`h→SetXTitle(“my x Axis“);`	Achsenbeschriftung
`h→SetAxisRange(10.,25.);`	Bereich ändern
`h→SetMarkerColor(kBlue);`	Symbolfarbe auf blau setzen
`h→SetMarkerSize(1.);`	Symbolgröße setzen
`h→SetMarkerStyle(20);`	Symbol setzen
`h→SetFillColor(kGreen);`	Histogrammfarbe
`h→SetFillStyle(3004);`	Diagonale Linien unter Histogramm
`h→Daw(“e“);`	Zeichnen, Einträge mit Fehler
`h→Print();`	Info am Bildschirm

Nach Änderungen mit obigen Methoden muss das Histogramm erneut gezeichnet werden oder das Canvas (z.B. c1) muss einen update erhalten.

```
c1 → Update();
```

Datenanalyse – erste Schritte

Wir wollen nun beispielhaft eine Messung analysieren, bei der das Ausgangssignal eines Verstärkers wiederholt gemessen wird. Der Messbereich ist [0,450]. Gleichzeitig wird mit jeder Messung die Raumtemperatur aufgezeichnet. Es werden 2 Textfiles geschrieben, signal.txt und temperature.txt

Arbeitsvorschlag:

- Verwenden Sie bitte als Arbeitsbasis das Programm analysis_frame.cc

- Schreiben Sie ein ROOT Macro, das das mittlere Verstärkersignal in verschiedenen Temperaturbereichen darstellt. Verwenden Sie zunächst die Methoden von TProfile. Passen Sie ein Polynom 1. Ordnung an und lesen sie die Fit Parameter aus (gleiche Verwendung wie bei der Gauss Anpassung, statt "gaus" jedoch "pol1") Diese sollen zur Temperaturkorrektur benutzt werden

analysis_1.cc

- Definieren Sie nun eigene nicht gleich große Temperaturbereiche und bestimmen Sie das mittlere Verstärkersignal mit einer Gauss Anpassung. Tragen Sie die Werte gegen die Temperatur auf. Wir wollen korrigierte und unkorrigierte mittlere Verstärkersignalwerte in der gleichen graphischen Darstellung haben. Passen Sie jeweils ein Polynom 1. Ordnung an.

analysis_2.cc

Datenanalyse – erste Schritte

Wir wollen nun beispielhaft eine Messung analysieren, bei der das Ausgangssignal eines Verstärkers wiederholt gemessen wird. Der Messbereich ist [0,450]. Gleichzeitig wird mit jeder Messung die Raumtemperatur aufgezeichnet. Es werden 2 Textfiles geschrieben, signal.txt und temperature.txt

Lösungsschritte:

- Das Programm analysis_frame.cc stellt die Daten im vector Temperature und Signal zur Verfügung.
- Erzeuge ein 2D Histogramm Temperature vs Signal. Wir müssen hier ein sinnvolles Binning für die Temperatur wählen. Mit der Methode ProfileX erhalten wir ein TProfile Objekt.
- Das TProfile Objekt kann mit der Methode Fit verwendet werden.
- Direkte Verwendung der Methode TProfile. Wie beim 2D Histogramm wird es mit der Methode Fill gefüllt. Auf das TProfile Objekt kann ebenfalls Fit angewendet werden.
- Die Darstellung erfolgt in beiden Fällen mit Draw().
- Der Wert der Anpassung für die Steigung dient zur Temperaturkorrektur
- Für den 2. Teil wird ein array mit Temperaturbereichen definiert.
- Initialisiere ein Histogramm Zeiger array und instanziere die Histogramme
- Diese werden gefüllt und mit einem Gauss gefittet.
- Mittelwerte und Fehler des Fits werden in arrays geschrieben.
- Die Temperaturbins und die halbe Breite werden ebenfalls in ein array geschrieben.
- Darstellung mit TGraphErrors.

ROOT - Funktionsklassen

TF1 Objekte in ROOT definieren 1 D Funktionen in einem Intervall. Auf eingebaute Funktionen, Polynome, Gaussverteilung, …. haben wir mit der `TH1::Fit` Methode über die Angabe des Funktionsnamens zugegriffen

"gaus"	$f(x) = p_0 \cdot e^{-\frac{1}{2}(\frac{(x-p_1)}{p_2})^2}$
"expo"	$f(x) = e^{p_0+p_1 x}$
"polN" N=0-9	$f(x) = p_0 + p_1 x + p_2 x^2 + ... + p_n x^n$
"chebychevN" N=0-9	$f(x) = p_0 + p_1 x + p_2(2x^2 - 1) + ...$
"landau"	$p(x) \approx \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(x+e^{-x})}$
"gausn"	$f(x) = p_0 \cdot e^{-\frac{1}{2}(\frac{(x-p_1)}{p_2})^2/(p_2\sqrt{2\pi})}$

- Möglichkeiten zur Definition von Funktionen

  Die Instanzierung von TF1 Objekten in ROOT erfolgt über verschiedene Konstruktoren

  - Formeln sind definiert als Character string (TFormula) ohne Parameter

```
TF1 * f1 = new TF1("f1","exp(-1.0*x)*sin(10.0*x)",0.,5.0);
```

Funktion | Funktionsbereich

ROOT - Funktionsklassen

- Möglichkeiten zur Definition von Funktionen

  - Formeln sind definiert als Character string (TFormula) mit Parametern

```
TF1 * f2 = new TF1("f2","[0]*exp([1]*x)*sin([2]*x)",0.,5.0);
f2->SetParameter(0,0.8);
f2->SetParameter(1,-1.0);
f2->SetParameter(2,9.0);
```

Funktion | Funktionsbereich

Parameter setzen

  - User definierte Funktionen (C++ code)

Variable x_0-x_n | Parameter p_0-p_n

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

1 Variable x_0, 3 Parameter p_0-p_2

  - User definierte Funktionen, Lambda Ausdruck (anonyme Funktion) (C++ code)

```
TF1 f1("f1","sin(x)",0,10);
TF1 f2("f2","cos(x)",0,10);
TF1 fsum("fs","[&](double *x, double *p){
                return p[0]*f1(x) + p[1]*f2(x); }",0,10,2);
fsum.SetParameters(0.7,0.3);
fsum.Draw();
```

Anzahl der Parameter

Funktionsobjekte in ROOT

- Funktionsdefinition in ROOT mit Funktoren

Beispiel:

```
#include <iostream>
#include <TMath.h>
#include <TF1.h>
struct myFunction{
      double operator()(double *x, double *p){
          return  p[0]*TMath::Gaus(x[0],p[1],p[2]);}
};
void myFunc() {
  myFunction f;
  int nParameter = 3;
  double xLow = .0 ; double xUp = 10.0 ;
  TF1 *gr = new TF1("gr",&f,xLow,xUp,nParameter);
  gr->SetParameters(10.,5.,0.5);
  gr->Draw();
}
```

FuncObject.C

ROOT - Funktionsklassen

- TF1 Beispiel: C++ Funktionen mit Variablen und Parameter

```
// Macro myFunc.C
Double_t myfunction(Double_t *x, Double_t *par)
{
   Double_t xx =x[0];
   Double_t f = TMath::Abs(par[0]*sin(par[1]*xx)/xx);
   return f;
}
void myFunc()
{
   TF1 *f1 = new TF1("myFunc",myfunction,0,10,2);
   f1->SetParameters(2,1);
   f1->SetParNames("constant","coefficient");
   f1->Draw();
}
```

Daten

Parameter p_0-p_n

Parameteranzahl

Wertebereich der Funktion

Setzen der Funktionsparameter

In der ROOT session

```
Root > .L myFunc.C
Root > myFunc();
```

ROOT - Funktionsklassen

- TF2 – 2D Funktionen

```
TF2 * f = new TF2("f","exp(-1.0*x)*sin(10.0*x)*cos(y)"
                                                ,0.,5.0,0.,6.3);
f->Draw("fsurf3")
```

- Mit Funktionen (TFn) können wir folgendes machen

Drawing	`f->Draw()`
Define plot range	`f->GetXaxis()->SetRangeUser(1.,10.)`
	`f->GetYaxis()->SetRangeUser(-3.,3.)`
Printing	`f->Print()`
Evaluate values	`f->Eval(1.7)`
Integration	`f->Integral(0.6,0.99)`
Differentiate	`f->Derivative(.2)`
Change line attributes	`f->SetLineColor(kRed)`
	`f->SetLineStyle(2)`
	`f->SetLineWidth(1)`
	….....

ROOT – Zufallszahlen

Bei der Modellierung von Prozessen spielen einem funktionalen Zusammenhang entsprechend verteilte Zufallszahlen eine große Rolle. In ROOT lassen sich Zufallszahlen mit verschiedenen Algorithmen generieren:

TRandom1 → RANLUX algorithm (3 ns pro call, nur zum Testen benutzen)
TRandom2 → Tausworthe generator of L'Ecuyer (5 ns pro call)
TRandom3 → (Mersenne-Twister Algorithmus, lange Periode von 2**19937 -1, 4.5 ns pro call).
TRandomMixMax → MIXMAX matrix generators (6-10 ns pro call)

Erklärungen und andere Algorithmen: https://root.cern.ch/doc/master/classTRandom.html

- Zufallszahlen in ROOT

Wir können gleichzeitig mehrere verschiedene Instanzen verwenden

```
#include <TRandom.h>
TRandom3 *R = new TRandom3();   // Instanzieren des Objektes
R→SetSeed(0);                   // set seed to machine clock,
                                // Zahl für feste Sequenz
Double_t MyRandom = R→Rndm();   // [0,1]Gleichvert. Zufallszahl
```

Ein `TRandom` Objekt kann in ein root File geschrieben werden

```
gRandom→Write("R");
```

ROOT – Zufallszahlen

- Zufallszahlen in ROOT

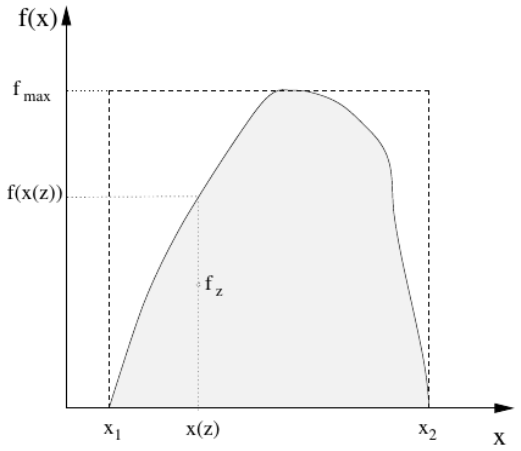
Weitere Mehtoden um Zufallszahlen gemäß von Verteilungen zu generieren:

```
UInt_t  i  = R→Integer(iMax);     //[0,iMax] UInt_t Zufallszahl
Double_t x = R→Uniform(x1,x2);    // [x1,x2] Zufallszahl
Double_t x = R→Gaus(mean,sigma); //gaussverteilte Zufallszahl
Double_t x = R→Exp(tau);          //exponentielle Zufallszahl
Double_t x = R→Poisson(mean);    //poissonverteilte Zufallszahl
Double_t x = R→Landau(mpv,sigma);//landauverteilte Zufallszahl
Double_t x = R→Binomial(ntot,prob);//binomialvert. Zufallszahl
void Sphere(x,y,z,r);             // generate random vectors
void Circle(x,y,r);               //x,y random 2D vector of
                                    length r
```

Beliebige Wahrscheinlichkeitsverteilungen

- Hit und Miss Methode

Neben dieser im ROOT Toolkit benutzbaren Funktionalität lassen sich Verteilungen auch mit Monte Carlo Methoden erzeugen.



Algorithmus:

- Erzeuge x aus $[x_1, x_2]$
- Erzeuge $z = f(x)$ aus $[0, f_{max}]$
- Akzeptiere x wenn $z < f(x)$

Beispiel für eine Funktion, die zufällige Werte aus dem unteren linken Quadranten des Einheitskreises zurückgibt.

HitUndMiss.cc

ROOT – Histogramme aus TF1

In ROOT lassen sich auch Histogramme mit Zufallszahlen füllen, die dem funktionalen Verlauf eines TF1 Objektes oder einem anderen Histogramm entsprechen.

- Histogramme mit Zufallszahlen aus TF1 Funktionen

  Es können sowohl die intern definierten Funktionen als auch beliebige TFN Funktionen zur Verteilung der Zufallszahlen verwendet werden.

```
TH1F *h = new TH1F ("myRandom","Histogramm from Landau",
                                                100,0.,10.);
h→FillRandom("landau",10000);
```

Funktion Zahl der Einträge

Oder

```
TF1 * f2 = new TF1("f2","[0]*exp([1]*x)*sin([2]*x)",0.,5.0);
f2->SetParameter(0,0.8);
f2->SetParameter(1,-1.0);
f2→SetParameter(2,9.0);

h→FillRandom("f2",10000);
```

myFunc.C

ROOT – Histogramme aus TF1

- Histogramme mit Zufallszahlverteilung aus anderen Histogrammen

  Liegen Messungen in Form von Histogrammen vor, können auch Histogramme mit Zufallszahlverteilungen bezüglich der Messungen generiert werden.

```
TH1F *myMeasure=new TH1F("myMeasure","Gemessenes Histogramm",
                                                    100,0.,100.);
TH1F *mySim=new TH1F("mySim","Simuliertes Histogramm",
                                                    100,0.,100.);
mySim→FillRandom(&myMeasure, 10000);
```

ROOT – TFn und Zufallszahlen

- Verteilung von Zufallszahlen gemäß einer TFn Funktion

Um den Einfluß von funktionalen Abhängigkeiten in Messungen zu untersuchen, ist häufig eine Verwendung von Zufallszahlverteilungen nützlich. TRandom enthält vordefinierte Verteilungen, es lassen sich aber auch Zufallszahlen bezüglich definierter TFn Funktionen generieren.

```
TF1 * f1 = new TF1("f1","[0]*exp([1]*x)*sin([2]*x)",0.,5.0);
f1→SetParameter(0,0.8);
f1→SetParameter(1,-1.0);
f1→SetParameter(2,9.0);

TRandom3 *R = new TRandom3();
R->SetSeed(0);  // set seed to machine clock
for (int I = 0 ; I < 100 ; I++) {
    Double_t gaussSignal = R->Gaus(averageSignal,sigma) ;
    Double_t f1Signal = f1->GetRandom();
}
```

Funktions- und Parameterdefinition

Gaussverteilte Zufallszahlen mit Mittelwert und Sigma

Zufallszahlen verteilt wie f1

ROOT – TFn und Zufallszahlen

- Beispiel Maxwell-Bolzmann Geschwindigkeitsverteilung

myMaxwell.cc

Neben den in ROOT definierten Wahrscheinlichkeitsverteilungen können wir beliebige Verteilungen erzeugen.

Funktionsdefinition

```
double maxwellfunc (double *x, double *p) {
  double xx = x[0];          //Definiere *x = x[0] zu xx
  double f = 4*Pi()*Power(Sqrt(p[0]/(2*Pi()*p[1]*p[2])),3)
             *xx*xx*Exp(-p[0]*xx*xx/(2*p[1]*p[2]));
  return f;
}
```

Parameter setzen

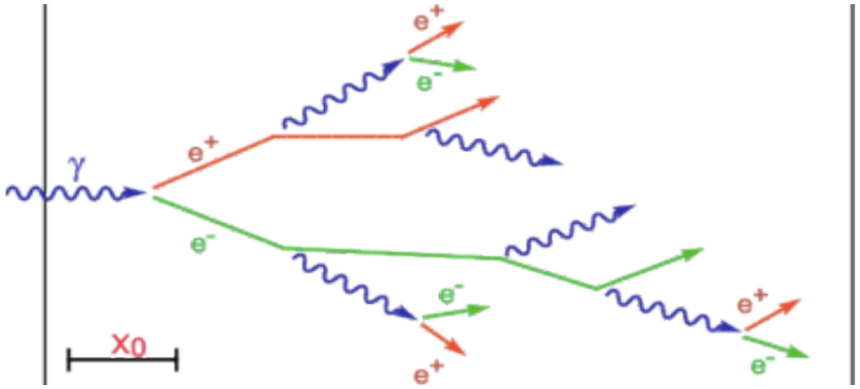
```
……..
  double k_B = 8.6173303e-5;       //Boltzmann Kostante in eV/K
  double m = 0.5109989461e-6;      //Masse eines Elektron in eV/c^2
  double T = 300.;                 //Temperatur des Teilchensystems

  TF1 *f1 = new TF1("f1",maxwellfunc,0.,1000.,3);
  f1→SetParameters(m,k_B,T);
```

Funktion erzeugen

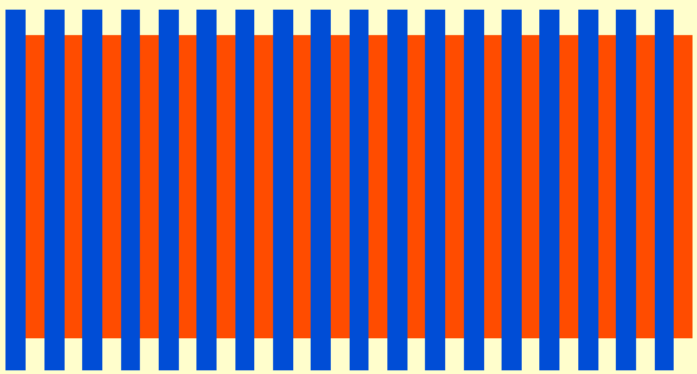
```
  TH1F *T300 = new TH1F("Maxwell Bolzmann T=300",3000,0.0,3000.0);
  T300→FillRandom("f1",5000);
…….
```

Zufallszahlen verteilt wie f1

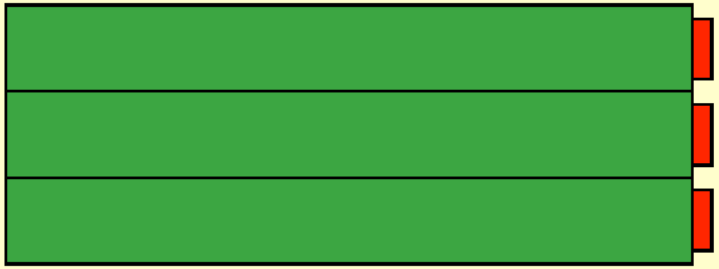
Energiemessung von Photonen



Sampling Kalorimeter



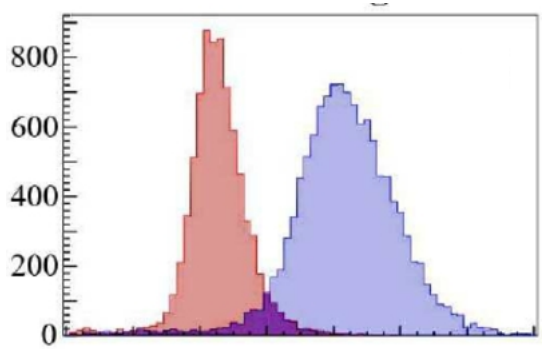
Homogene Kalorimeter



Beiträge zur Energieauflösung:

Sampling / Quantum Fluktuationen	$\sigma_s/E \sim \frac{a}{\sqrt{E}}$	$a = 10\%$
Elektronik Noise	$\sigma_N/E \sim \frac{b}{E}$	$b = 0.1\,GeV$
Struktur / Aufbau	$\sigma_c/E \sim c$	$c = 0.35\%$

$$\sigma_{tot}^2 = \sigma_s^2 + \sigma_N^2 + \sigma_c^2$$



Simulation von Verteilungen

Die Energieauflösung σ/E eines elektromagnetischen Kalorimeters zur Energiemessung von Photonen besteht aus 3 Anteilen mit verschiedenen Energieabhängigkeiten. Die gesamte Energieauflösung ist die quadratische Summe der einzelnen Anteile.

Arbeitsvorschlag:

- Schreiben Sie jeweils eine TF1 Funktion der Kalorimeterauflösung für sigma und eine für sigma/E als C++ Funktion: `Double_t sigma(Double_t *x, Double_t *par);`
  `Double_t sigmaoverE(Double_t *x, Double_t *par);`
  mit
  ```
  Double_t sampling =  par[0];
  Double_t noise   =   par[1] ;
  Double_t constant  = par[2] ;
  ```
  Zeichnen Sie beide in ein Canvas für 1-150 GeV.
- In das Canvas für Sigma/E soll der funktionale Verlauf der einzelnen Anteile eingezeichnet werden. Implementieren Sie dazu separate TF1 Funktionen der Anteile.
- Simulieren Sie die gemessene Kalorimeterenergie für 6 feste Energiewerte und tragen Sie die Werte jeweils in ein Histogramm ein. Pro Energiewert wollen wir 1000 Werte simulieren.

CaloResolution.C

ROOT – Input / Output

I/O ist für ein Datenanalyse Tool von großer Bedeutung. Mit ROOT lassen sich, im Gegensatz zu nativem C++, Objekte in Files schreiben.

Was wird gebraucht um Objekte in Files / Disks zu speichern (Serialization) ?

- Typ des Objekts zur Laufzeit → runtime type information (RTTI)
- Laufzeitinfos des Objektes (Typ, Größe, Member) → reflection
- Ort der Datenmember des Objektes im Speicher → introspection
- Lesen des Datenstroms aus dem Speicher auf die Disk → raw I/O

→ komplexe Aufgabe ohne vollständige native Unterstützung in C++.

Ziel: I/O mit einer Rate von einigen 10 PByte Daten / Jahr

ROOT verwendet ein komprimierendes (gzip) maschinenunabhängiges binäres Format, das Daten und die Beschreibung (Dictionary) in ROOT Files speichert. Die ROOT Files besitzen eine Directory Struktur. Das Dictionary mit reflection data aller Typen im Quellcode wird automatisch erzeugt und in Files *.pcm gespeichert, Beispiel: Benutzung von ACLiC

```
Root[0] .L analysis.cc+
                          → analysis_cc_ACLiC_dict_rdict.pcm
```

https://root.cern.ch/root/htmldoc/guides/users-guide/InputOutput.html
https://root.cern/manual/io/

ROOT – Input / Output

Speichern von Objekten in Files mit ROOT

- Öffnen eines TFiles

```
TFile * f = TFile::Open("myFile.root","RECREATE");
```

- Schreiben einer Instanz von `TObject`

```
object→Write("OptionalName");
```

mit `"OptionalName"` oder `TObject::GetName()`

- Schreiben eines beliebigen Objektes bei bekanntem dictionary

```
f→WriteObject(object,"Name");
```

TFile – Klasse zum Schreiben von ROOT Files

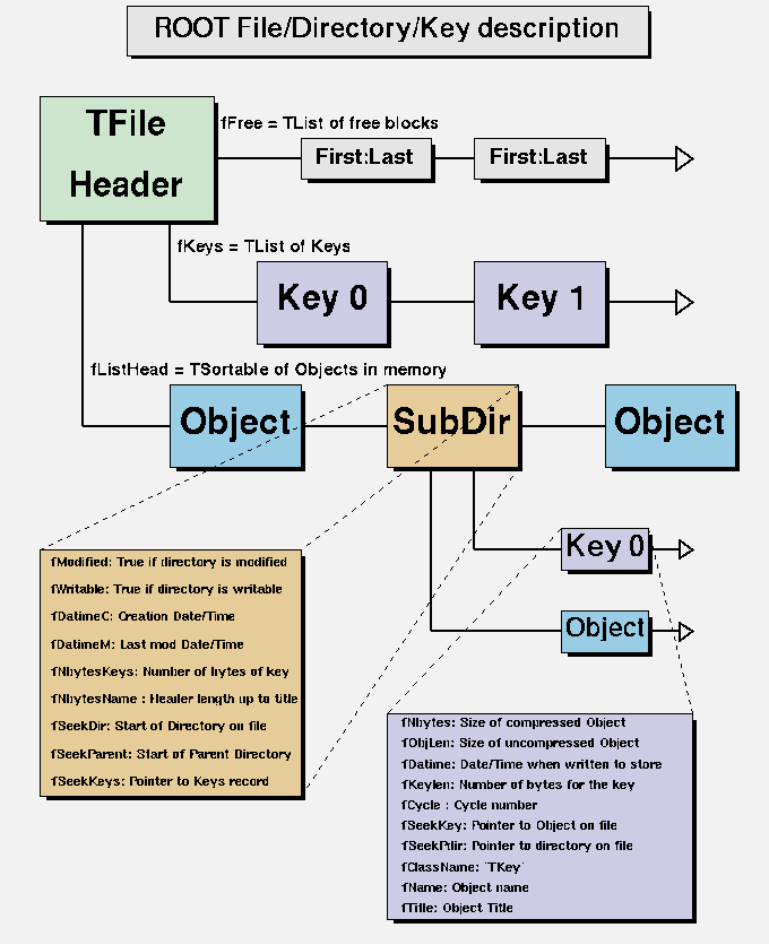
NEW
RECREATE
READ
UPDATE

```
TFile * myFile = new Tfile ("myFile.root","NEW");
if ( myFile→IsOpen() ) cout << "File openend" << endl;
```

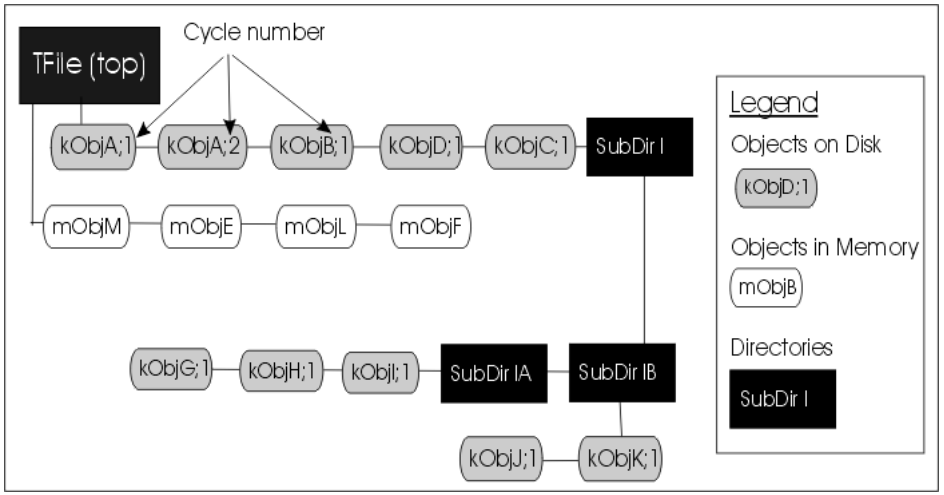
`myFile` wird Default für den gesamten I/O (setzt die globale Variable `gFile`)

```
myFile→Close();

delete myFile;
```

ROOT – Input / Output



ROOT File Structure



TFile Structure

ROOT – Input / Output

TFile – Directory

ROOT files besitzen eine directory Struktur. Es gibt 2 globale Variable `gFile` und `gDirectory`, die auf das aktuelle TFile und das aktuelle directory zeigen. Die globale Variable `gROOT` zeigt auf das top level ROOT Objekt TROOT.

```
root [0] TFile *f1 = new TFile("AFile1.root","NEW");
root [1] gDirectory→pwd();                    // das neue TFile wird aktuelles directory
AFile1.root:/
root [2] f1→mkdir("myStuff");                 //directory myStuff wird erzeugt
root [3] gDirectory→pwd();
Afile1.root:/
root [4] f1→rmdir("myStuff");                 //directory myStuff wird gelöscht
AFile1.root:/
root [5] f1→cd("myStuff");                    // Wechsel in das directory myStuff
root [6] gDirectory→pwd();
Afile1.root:/myStuff
root [7] f1→GetCompressionFactor();           // Kompressionsfaktor
(float) 1.00000f
root [8] .ls
TDirectoryFile*   myStuff myStuff
root [9] gROOT→cd();                          // CLing wird aktuelles directory
root [10] gDirectory→pwd();
Rint:/
```

ROOT – Input / Output

TFile – Write/Read

```
root [10] TH1F*h=new TH1F("h","Interactive;X;Entries",100,-5,5);
root [11] h->FillRandom("gaus");      // create and fill histogramm h
root [12] f1→cd();                    // goto Tfile f1
root [13] h→Write();                  // write histogramm to File
root [14] gDirectory→ls("-d");        // content on disk
TFile**            AFile1.root
TFile*             Afile1.root
TDirectoryFile*    myStuff myStuff
KEY: TDirectoryFile myStuff;1
KEY: T H1F          h;1   Interactive
root [15] f1→GetCompressionFactor()   // Wechsel in das directory myStuff
(float) 1.99421f
root [16] gDirectory→ls("-m");        // see content in memory
root [17] h→Draw();                   // Draw erzeugt ein Canvas
Info in <Tcanvas::MakeDefCanvas>:create default canvas with name c1
root [18] c1→Write();                 // Write Canvas to File

root [19] vector <int> p{2,5,7};                  // create int vector p
root [20] gDirectory→WriteObject(&p,"p_1");       // Write vector p to the root file
root [21] vector <int> *l;                        // create pointer of int vector l
root [22] gDirectory→GetObject("p_1",l);          // copy vector back to l
root [23] cout << l->size() <<endl;
```

ROOT – Input / Output

TFile – Write/Read

```
root [24] TH1F *hc=(TH1F*)gDirectory→Get("h");    // Read histo h from File
root [25] TFile*f3 = Tfile::Open(     // Remote acces of TFile
          "https://www.physi.uni-heidelberg.de/~marks/analysis.root");
root [25] gDirectory→pwd();
https://www.physi.uni-heidelberg.de/~marks/analysis.root:/
root [25] gDirectory→ls();
TWebFile**              https://www.physi.uni-heidelberg.de/~marks/analysis.root
 TWebFile*              https://www.physi.uni-heidelberg.de/~marks/analysis.root
  KEY: TH1D     S;1     Signal
  KEY: TH1D     T;1     Temperature
  KEY: TH2D     C2;1    Temperature vs Signal
  KEY: TCanvas  myC;1   Signal Plots
  KEY: TH1D     S1;1    Signal 19<T<20
  KEY: TH1D     S2;1    Signal 20<T<21
  KEY: TH1D     S3;1    Signal 21<T<22
  KEY: TH1D     S4;1    Signal 22<T<23
  KEY: TCanvas  myTempDep;1     T dependent Signal Plots
  KEY: TH2D     CS2;1   Temperature vs Signal corrected
  KEY: TH1D     CS;1    Signal corrected
  KEY: TH1D     CSCLONE;1       Signal corrected
  KEY: TH1D     BCK;1   background
  KEY: TH1D     CSSUB;1 Signal corrected, Bck subtracted
  KEY: TCanvas  myCor;1 T corrected Signal Plots
```

rootIO_commands.txt

ROOT – Input / Output

TFile – Write/Read to XML file format

```
root [26] TFile *f = TFile::Open("Example.xml","recreate");// Open XML
root [27] TH1F *h = new TH1F("h","test",1000,-2,2)
root [28] h->FillRandom("gaus");
root [29] h->Write();

root [30] TFile *f = Tfile::Open("Example.xml"); // Open XML for reading
root [31] TH1F *h = (TH1F*)f→Get("h");           // Read histo h from File
root [32] h->Draw();
```

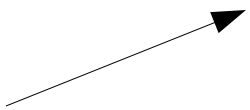
Objekte in ROOT Files

Wir können nicht nur Instanzen von `TObject` in ROOT Files schreiben/lesen, sondern auch beliebige Klassenobjekte. Dazu werden Run Time Type Information (RTTI) benötigt, d.h. die Fähigkeit einer Klasse sich selbst zu analysieren.

In computer science, reflection is the ability of a computer program to examine, introspect, and modify its own structure and behavior at runtime

In ROOT ist reflection über TClass implementiert. Hier wird Information über die Klassen, Methoden und Daten, Kommentarfelder und Parametertypen zur Verfügung gestellt. Dies erfolgt mit dem `ClassDef` macro

```
#include <TDirectory.h>
class MyClass
{
   public:
   …..
   ClassDef(MyClass,ClassVersionID);
   …..
}
```



`ClassVersionID ≥ 0`
Erlaubt eine Versionskontrolle die vom ROOT I/O benutzt werden kann

und der Erzeugung einer reflection database, einem ROOT dictionary. Für die Klasse muss ein default constructor (constructor ohne Parameter oder mit Parameter aber gesetzten default Werten) existieren.

Objekte in ROOT Files

- ROOT dictionaries

  Dictionaries werden in CLing automatisch mit ACLiC erzeugt

  ```
  root> .L MyHeader.h+
  ```

  + ruft automatisch einen Generator, der ein dictionary File, ein pcm File und eine shared library erzeugt.

  Für generelle C++ Programme werden dictionaries mit Hilfe von preprocessor link Definitionen, die in speziellen Files `LinkDef.h` untergebracht sind, und mit dem class header mit dem tool `rootcling` erzeugt.

  ```
  rootcling -f DictOut.cxx -c OPTIONS header.h … Linkdef.h
  ```

  Das File `DictOut.cxx` enthält Streamer() und ShowMembers() Methoden unserer Klasse. Dabei wird Streamer() benutzt um ein Objekt in/aus TBuffer zu streamen and ShowMembers() wird von den Dump() und Inspect() Methoden von TObject verwendet.
  Das File `LinkDef.h` definiert, welche Klasse wie zum Dictionary File hinzugefügt werden sollen.

  ```
  #ifdef __CLING__
  #pragma link C++ class MyClass+ ;          // + improves I/O
  #endif
  ```

Objekte in ROOT Files

Es können auch komplexe preprocessor Anweisungen verwendet werden

```
#ifdef __CLING__
#pragma link off all functions;                 Abschalten aller Funktionen
#pragma link C++ function f;                    Hinzufügen von f und g
#pragma link C++ function g(int,double);
#pragma link C++ MACRO max;                     Macro Definition
#pragma link C++ class A+;                      Hinzufügen einer Klasse
#pragma link off function A::h(double);         Abschalten einer Methode
#endif
```

- Erzeugen von shared libraries

  Um die Klasse MyClass in einer Anwendung verwenden zu können, wird sie in Form einer shared library hinzugefügt, die auch die dictionary Information enthält

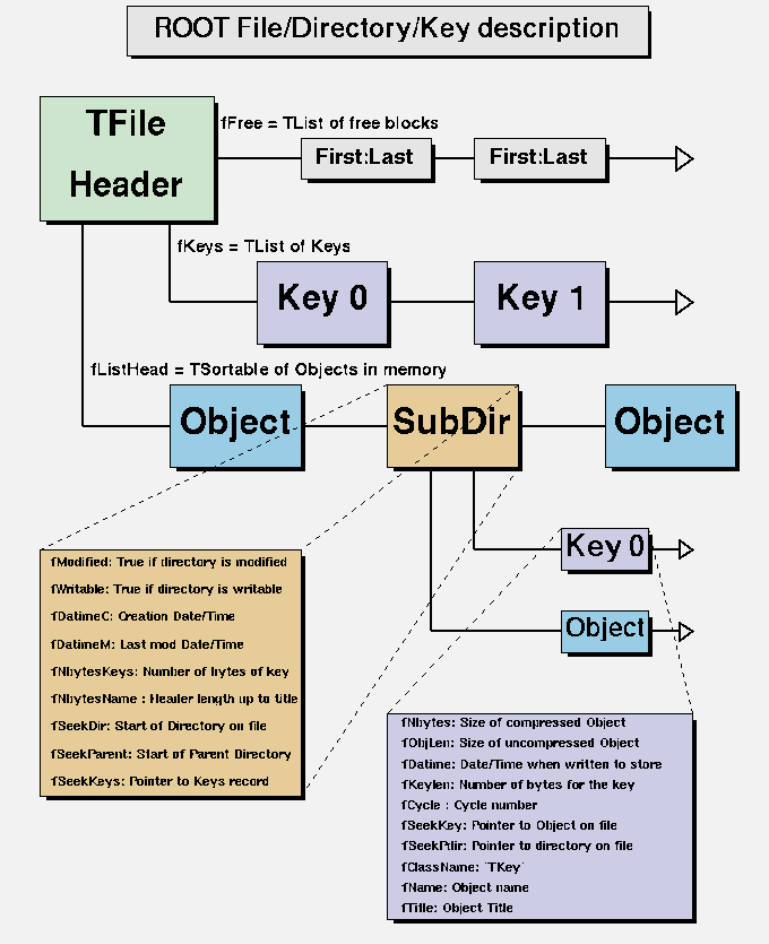
```
gcc -shared -fPIC -o libMyClass.so \
    `root-config --cflags --ldflags --glibs` \
    DictOut.cxx MyClass.cc
```

  Das File DictOut_rdict.pcm muss dabei im directory der shared library existieren.

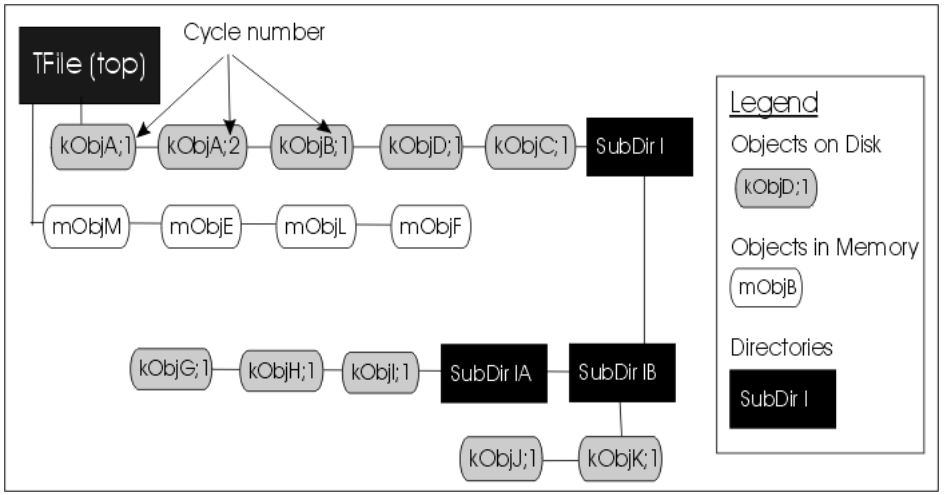
- Erzeugen einer C++ Anwendung

```
g++ myApp.cc -L. -lMyClass -o myApp  \
 `root-config --ldflags --glibs --cflags `
```

ROOT – Input / Output



ROOT File Structure



TFile Structure

```
root [0] TFile *f1 = new TFile("signal.root","READ");   // Lesen des TFile
root [1] f1->ls();                                      // List TFile Information
root [2] f1->Map();                                     // record address, bytes im record, class name, comp. factor
root [3] f1->ShowStreamerInfo();                        // Streamer Info
```

ROOT – Speichern von Objekten

Alle Objekte die von TObject erben können in ein ROOT- File geschrieben werden (persistent Object). Die Objekte können anschliessend wieder eingelesen werden.

Beispiel Schreiben:

Beispiel in myRooFileWrite.C

```
…...
void myRooFileWrite{
      TRandom3 *R = new TRandom3();
      R->SetSeed(0);
      TH1F *h = new TH1F ("myhist","Write Test",100,-5.,5.);
      h->FillRandom("gaus",10000);
      TFile outFile ("myRooFileWrite.root","RECREATE");
      h->Write();
      outFile.Close();
      return;
}
```

```
>$ root myRooFileWrite.root
root[0] _file0->ls()
TFile**         myRooFileWrite.root
TFile*          myRooFileWrite.root
KEY: TH1F     myhist;1        Write Test
myhist->Draw()
```

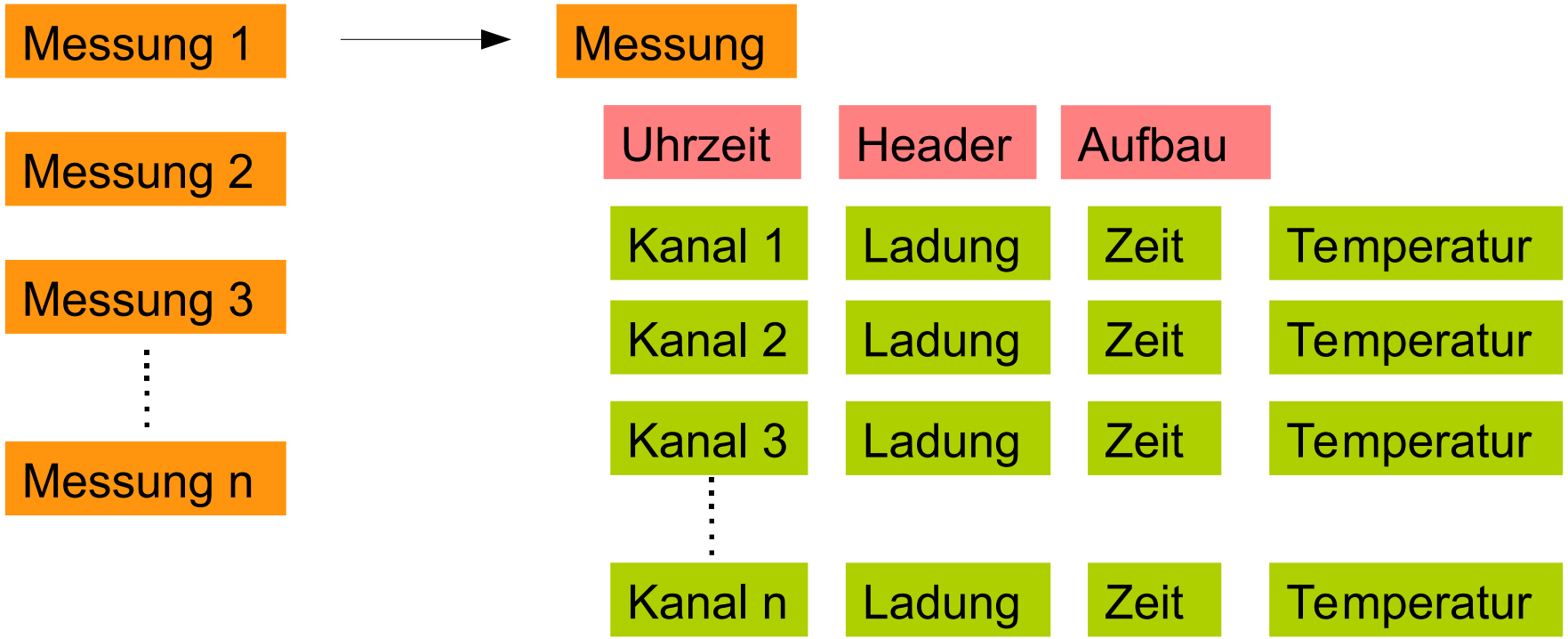
ROOT File öffnen

Inhalt auflisten

Histogramm darstellen

TTree - Speichern von Daten

Ein ROOT Tree enthält eine Daten Struktur, die in Form einer C++ Klasse dargestellt werden kann. Die Struktur wird dabei wiederkehrend in Form von Einträgen gespeichert. Der gesamte Tree läßt sich in einem ROOT File speichern und kann anschliessend wieder gelesen werden.



Der Tree ist in Äste (branches) unterteilt, die die Objekte speichern und leaves, die den Datenzugang erlauben. Spezielle Techniken sorgen für effizienten Input/Output.

TTree - Speichern von Daten

TTree kann als array einer Datenstruktur im Speicher aufgefasst werden. Die tree entries (Variablen) werden über ein mapping der Adressen bekannt gemacht und sequenziell geschrieben oder gelesen. Die branches (TBranch) sind Spalten im tabellenartigen tree. Die Daten werden in variablen festlegbaren Puffergrössen gespeichert (basket) und komprimiert in das ROOT File geschrieben. Informationen über die trees werden über header gespeichert und erlauben schnellen IO.

Es lassen sich beim Lesevorgang nur benötigte Teile eines branches lesen. Ebenso können subsets der TTree Einträge gelesen werden.

Trees können durch zusätzliche branches mit friend trees TTree::AddFriend()) erweitert werden.

Daten lassen sich auch über viele ROOT Files verteilen und können mit TChain wieder zu einem einzigen Tree zusammengefügt werden.

Trees lassen sich mit Ausdrücken indizieren, die von Werten in den leaves abhängen, z.B. Run Nummer und Ereignis Nummer

TTree - Ntuple

- Ein Beispiel mit einer tabellarischen Datenstruktur stellt das Ntuple dar.

Im Beispiel myRooFileWriteText.C wird ein Text File mit 4 Spalten geschrieben. Dieses Text File wird dann mit myRooFileReadText.C in ein Ntuple geschrieben und in einem root File gespeichert.

TNtuple holds floating point numbers only

```
Float_t px,py,pz,p,E;
TFile *f = new TFile("myRooNtuple.root","RECREATE");
TNtuple *ntuple = new TNtuple("ntuple",
                              "data from asci File","px:py:pz:p:E");
for (int i = 0 ; j < nRead ; j++)
                     ntuple→Fill(px,py,pz,p,E);
f->Write();
```

Mit Hilfe des TBrower Werkzeugs kann das Ntuple im root File dann angesehen werden und auch am ROOT prompt manipuliert werden.

ROOT File öffnen

Inhalt ansehen

```
>$ root myRooNtuple.root
root[0] Attaching file myRooNtuple.root as _file0...
root[1] TBrowser T
root[2] _file0→ls()
…..
root[3] TCanvas *myCanvas = new Tcanvas()
root[4] ntuple->Draw("px:E")
root[5] ntuple→Draw("px:E","p>600")
```

Neues Canvas erzeugen

Plot px vs E für p>600

TTree - Beispiel

- Schreiben eines TTree Objektes

Im Beispiel myWriteTTree.C wird ein TTree Objekt, das 1000 Elemente der Größen E und des arrays p[3] enthält, in einem ROOT File gespeichert.

```
…..
TFile *f = new Tfile("myTTree.root","RECREATE");
TTree *t = new TTree("tree","data tree");

int nEvent;     t->Branch("nEvent",&nEvent,"nEvent/I");
double E;       t->Branch("E",&E,"E/D");
double p[3];    t->Branch("p",p,"p[3]/D");

for (int i = 0 ; i < 1000 ; i++){
   nEvent++;
   E = …
   p[0] = ….
   t→Fill();
}

f->cd();
t->Write();
f->Close();
}
```

Specify TTree name

In einen Branch können beliebig komplexe Objekte geschrieben werden.

TTree - Beispiel

- Schreiben eines TTree Objektes

Im Beispiel myWriteTTree.C wird ein TTree Objekt, das 1000 Elemente der Grössen E und des arrays p[3] enthält, in einem ROOT File gespeichert.

```
…..
TFile *f = new TFile("myTTree.root","RECREATE");
TTree *t = new TTree("tree","data tree");

int nEvent;     t->Branch("nEvent",&nEvent,"nEvent/I");
double E;       t->Branch("E",&E,"E/D");
double p[3];    t->Branch("p",p,"p[3]/D");

for (int i = 0 ; i < 1000 ; i++){
   nEvent++;
   E = …
   p[0] = ….
   t→Fill();
}

f->cd();
t->Write();
f->Close();
}
```

Specify TTree name

In einen Branch können beliebig komplexe Objekte geschrieben werden.

TTree - Beispiel

- Lesen eines TTree Objektes

Im Beispiel myReadTTree.C wird das im vorherigen Beispiel geschriebene TTree Objekt gelesen.

```
…..
TFile *f = new TFile("myTTree.root");
TTree *t = (TTree*) f->Get("tree");

TBranch *b_nEvent;
TBranch *b_E;
TBranch *b_p;

int nEvent;  t->SetBranchAddress("nEvent",&nEvent,&b_nEvent);
double E;    t->SetBranchAddress("E",&E,&b_E);
double p[3]; t->SetBranchAddress("p",p,&b_p);

Long64_t nentries = t->GetEntries();
for (Long64_t i=0;i<nentries;i++) {
    t->GetEntry(i);
    cout << "Event " << nEvent << "  p[0] = " << p[0] << endl;
}
```

Specify TTree name

Um I/O Funktionen wie das Schreiben von beliebigen Objekten in ROOT verwenden zu können, müssen wir

- das `ClassDef` macro in unsere Klasse hinzufügen
  (`#include TDirectory.h`)
- ein ROOT Dictionary mit `rootcling` erzeugen
- eine shared library bauen
- eine Anwendung erzeugen

Arbeitsvorschlag:

- Verwenden Sie unsere Klasse FourVector um das Schreiben von FourVector Objekten in ROOT Files zu testen.

- Schreiben Sie mehrere FourVector Objekte in einen ROOT Tree

- Lesen Sie den Tree wieder ein

readwriteObject.tar

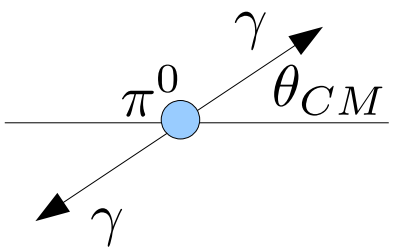
TTree - Beispiel

- TTree – Interactive Commands

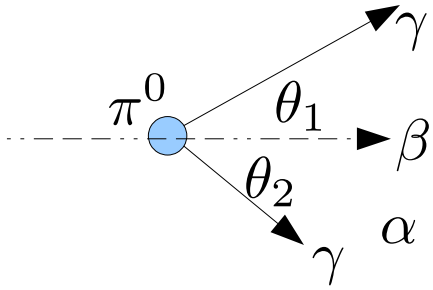
```
>$ root
root[0] TFile *f=new TFile("myTTree.root")  // open file with tree
root[1] TTree *a=(TTree *)f->Get("tree")// read tree
root[2] a->Print()                          // print summary of the tree
root[3] a->Show(12)                         // print entry number 12
root[4] a->Scan("E:p[1]","p[2]>0.5")        // select variable with conditions
root[5] a->Draw("p[1]:p[2]")                // Draw tree variables
root[6] a->StartViewer()                    // Start tree viewer gui
```

Simulation des π^0 Zerfalls

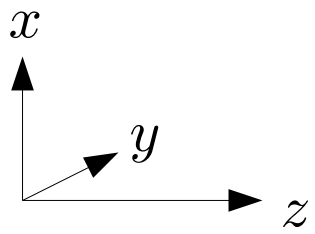
Schwerpunktsystem:



Laborsystem:



$$\alpha = \theta_1 + \theta_2$$



Relativistische Variablen:

$$\beta = \frac{|\vec{p}|}{E} \qquad \gamma = \frac{E}{m_0 c^2}$$

Photonen:

$$E_1^{lab} = \gamma \frac{m}{2}(1 + \beta cos(\theta_{CM}))$$

$$E_2^{lab} = \gamma \frac{m}{2}(1 - \beta cos(\theta_{CM}))$$

Invariante Masse

$$m^2 = 2E_1^{lab}E_2^{lab}(1 - cos(\alpha))$$

$$tan(\theta_1) = \frac{sin(\theta_{CM})}{\gamma(\beta + cos(\theta_{CM}))}$$

θ_{CM} wird im Bereich [0,pi] uniform verteilt generiert, daraus lassen sich die Photonenergien und Winkel bestimmen. Die π^0 Energie wird gegeben. Eine Abhängigkeit in der x-y Ebene ist nicht vorhanden.

Implementiert in pi0Decay.C

Simulation des π^0 Zerfalls

Die maximale und minimale Energie im Laborsystem ist:

$$E_{max}^{lab} = \gamma \frac{m}{2}(1+\beta) \qquad E_{min}^{lab} = \gamma \frac{m}{2}(1-\beta)$$

Der minimale Öffnungswinkel ist: $\alpha_{min} = arcsin(\frac{1}{\gamma})$

Die Grenzen der Energiewerte können mit den obigen Gleichungen implementiert werden.

Durch den Messprozess werden die Energiewerte und Winkel mit einer Unsicherheit gemessen, die durch die Auflösung des Detektors gegeben ist. Zur Simulation des Messprozesses nehmen wir an, das die Auflösungsfunktionen gaussverteilt und energieabhängig sind. Es werden folgende Funktionen benutzt:

$$\sigma_E = \sqrt{(a \cdot \sqrt{E})^2 + (c \cdot E)^2 + b^2}$$

$$\sigma_\theta = \frac{t}{\sqrt{E}}$$

Typische Werte:

$$a = 0.05\,\sqrt{GeV}$$
$$b = 0.07\,GeV$$
$$c = 0.01\,GeV^{-1}$$
$$t = 0.004\,mrad \cdot \sqrt{GeV}$$

Simulation des Zerfalls $\pi^0 \rightarrow \gamma\gamma$

Im Programm pi0Decay.C ist ein Zerfall $\pi^0 \rightarrow \gamma\gamma$ simuliert. Aus dem generierten Streuwinkel im Schwerpunktsystem werden die Photonenergien und Streuwinkel im Laborsystem berechnet und die Vierervektoren der Photonen erzeugt.

Arbeitsvorschlag:

- Es sollen für eine beliebige Anzahl von Zerfällen die beschreibenden Variablen in ein nTuple mit dem Namen “pi0Gen“ geschrieben werden. Gehen Sie dazu vom oben genannten Programm aus. Stellen Sie dann Variablen des ntuples mit TBrowser dar.

pi0DecaySim.C

- Ergänzen Sie ihr Programm, in dem Sie die Messung des generierten Zerfalls simulieren. Dazu werden die generierten Werte der Photonenergie und der Photon-winkel mit Auflösungsfunktionen (CaloFunc.C) versehen. Schreiben Sie die generierten Grössen und die gemessenen Photonvierervektoren in einen TTree. Vergleichen Sie die gemessene und die generierte invariante Masse.

pi0DecayCalo.C

Conclusion

Mit diesem Einblick in die Grundlagen eines modernen Analysewerkzeuges sollten Sie in der Lage sein

- im Selbststudium C++ Kenntnisse für komplexere Probleme zu erarbeiten
- die ROOT Documentation and Tutorials zu verstehen

Ausblick

Weiterführende, für eine Datenanalyse wichtige Themen, sind

- Datenanpassung mit Hilfe von ROOT/Minuit
  - Bestimmung von Modelparametern
- Datenanpassung mit hilfe von rooFit und rooStat
  - Statistische Methoden der Datenauswertung
- Multivariate Analysis
  - Multvariate Analyse zur Datenselektion und Untergrundreduktion mit dem ROOT Paket TMVA
- Neural networks
  - Artificial neural networks (ANN) mit der ROOT Klasse TmultiLayerPerceptron
  - Neural Network Objects (NNO)