

ROOT - Einleitung

Der Vergleich von Messungen mit theoretischen 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.

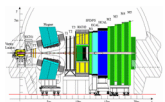
- 10 Vollzeit-Entwickler am CERN
- Offener Quellcode → software wird von den Usern mit- und weiterentwickelt
- Dokumentation, Tutorials und Quellcode: <https://root.cern.ch/drupal/>

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 21 member states from Europe with others associated
- ~ 3500 employees and about 10000 guest scientists
- The annual budget is about 750 million Euro
- The facility has 3 accelerators, PS, SPS and the LHC





SUISSE
FRANCE

CMS

LHCb

CERN Prévessin

ATLAS

CERN Meyrin

SPS 7 km

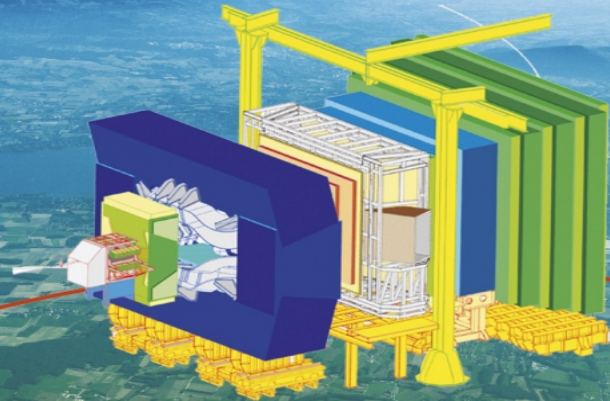
LHC 27 km

MS 4.33 m



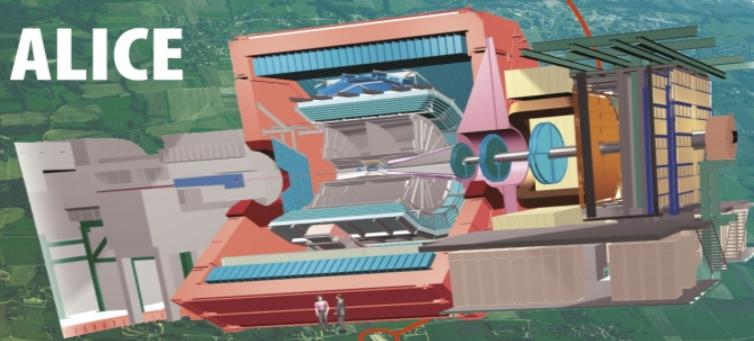
ATLAS

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

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

ROOT startup im Terminal Fenster mit

```
$> root
root [0] _
root [1] .?
root [2] .q
root [3] qqqqqq
```

ROOT history:
\$HOME/.root_hist

- root prompt versteht C++ mit dem Interpreter CINT (V5.xx) / Cling (V6.xx)
Python binding vorhanden
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 – Dokumentation

ROOT a Data a... x +

https://root.cern.ch


CERN / Moore/trunk/L0/L0App/... PIOrganization C++ / ROOT URZ Netzinfrastruktur - Rech... LHCb Meetings - Event... DB BAHN - Meine Bahn Leo Google Calendar

ROOT


Data Analysis Framework

Google™ Custom Search


Download **Documentation** News Support About Development Contribute




Getting Started



Reference Guide



Forum




Gallery

ROOT is ...

A modular scientific software framework. It provides all the functionalities needed to deal with big data processing, statistical analysis, visualisation and storage. It is mainly written in C++ but integrated with other languages such as Python and R.

Try it in your browser! (Beta)

 Download ROOT [or Read More ...](#)

Under the Spotlight

05-09-2016 [Get the most out of the ROOT tutorials!](#)

All ROOT tutorials are now available as ROOTBooks which can be statically visualized via [NBViewer](#) or interactively explored with [SWAN](#).

```
auto f = TFile::Open("events.root");
TTreeReader myReader("events",f);

using fourVectors = vector<ROOT::Math::PxPyPzEVector>;
TTreeReaderValue<fourVectors> tracksRV(myReader, "tracks");

while(myReader.Next()){
    auto tracks = *tracksRV;
    for (auto&& track : tracks)
        cout << "Track Pt is " << track.Pt() << endl;
}
```

[Previous](#) [Pause](#) [Next](#)

Other News

16-04-2016 [The status of reflection in C++](#)

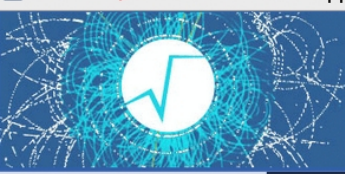
05-01-2016 [Wanted: A tool to 'warn' user of inefficient \(for I/O\) construct in data model](#)

ROOT: ROOT R... x +

https://root.cern.ch/doc/v608/

Search

CERN / Moore/trunk/L0/L0App/... PIOrganization C++ / ROOT URZ Netzinfrastuktur - Rech... LHCb Meetings - Event... DB BAHN - Meine Bahn Leo Google Calendar



ROOT 6.08/03
Reference Guide

ROOT Home Page **Main Page** Tutorials User's Classes Namespaces All Classes Files Release Notes Search

ROOT Reference Documentation

Introduction

Welcome to ROOT

This documentation describes the ROOT interface (API). This is not a direct copy from the source code but is derived directly from the source code at the page heading. You may find more information at the page heading.

How to use this reference

The [User's Classes](#) in the top navigation bar lists all the classes, both for the public and private C++ namespace can be found in the [User's Classes](#) section.

ROOT provides other 1

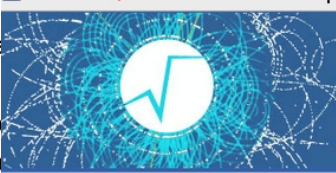
- A general [Users Guide](#)
- A number of topical [User's Classes](#)

ROOT: User's C... x +

https://root.cern.ch/doc/v608/modules.html

Search

CERN / Moore/trunk/L0/L0App/... PIOrganization C++ / ROOT URZ Netzinfrastuktur - Rech... LHCb Meetings - Event... DB BAHN - Meine Bahn



ROOT 6.08/03
Reference Guide

ROOT Home Page Main Page Tutorials **User's Classes** Namespaces All Classes Files Release Notes

User's Classes

Here is a list of all modules:

▼ Core ROOT classes	The Core classes of ROOT
Base ROOT classes	The Base classes of ROOT
Containers	The containers and generators the ROOT framework offers
▼ Geometry	The Geometry related packages
Geometry classes	The Geometry related classes
Geometry builder	The Geometry builder related classes
Geometry painter	The Geometry painter, checker, overlap and track related classes
▼ Graphics	The graphics related classes
▶ Graphics' Backends	Graphics' Backends interface classes
▶ 2D Graphics	The 2D graphics related classes

[ROOT Home Page](#)[Main Page](#)[Tutorials](#)[User's Classes](#)[Namespaces](#)[All Classes](#)[Files](#)[Release Notes](#)[Class List](#)[Class Index](#)[Class Hierarchy](#)[Class Members](#)

Class List

Here are the classes, structs, unions

- ▶ [N](#) BidirMMapPipe_impl
- ▶ [N](#) cling
- ▶ [N](#) Memstat
- ▶ [N](#) PyROOT
- ▶ [N](#) Rgl
- ▼ [N](#) RooStats
 - ▶ [N](#) HistFactory
 - [C](#) AcceptanceRegion
 - [C](#) AsymptoticCalculator
 - [C](#) BayesianCalculator

TSpectrum.cxx

Go to the documentation of this file.

```

1 // @(#)root/spectrum:$Id$
2 // Author: Miroslav Morhac 27/05/99
3
4 #include "TSpectrum.h"
5 #include "TPolyMarker.h"
6 #include "TVirtualPad.h"
7 #include "TList.h"
8 #include "TH1.h"
9 #include "TMath.h"
10
11 /** \class TSpectrum
12     \ingroup Spectrum
13     \brief Advanced Spectra Processing
14     \author Miroslav Morhac

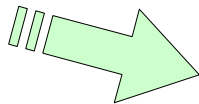
```

ROOT source code

Advanced spectra processing functions for:

- 18 - One-dimensional background estimation
- 19 - One-dimensional smoothing
- 20 - One-dimensional deconvolution
- 21 - One-dimensional peak search
- 22

ROOT Class Index



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");
```

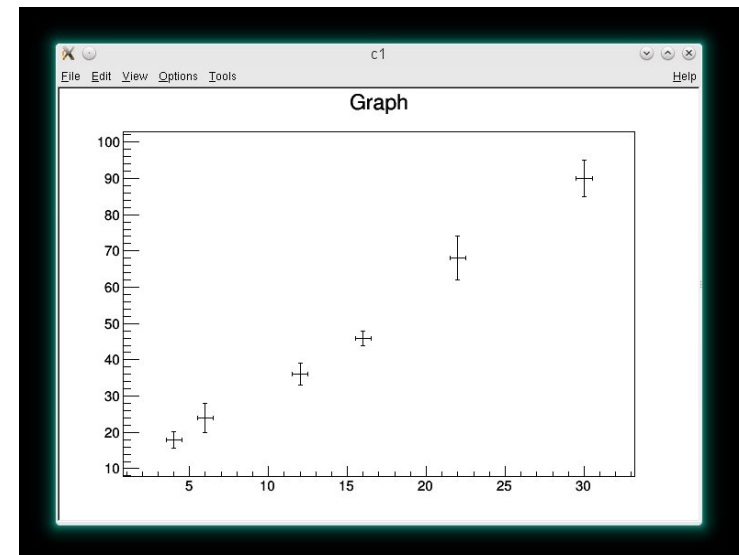
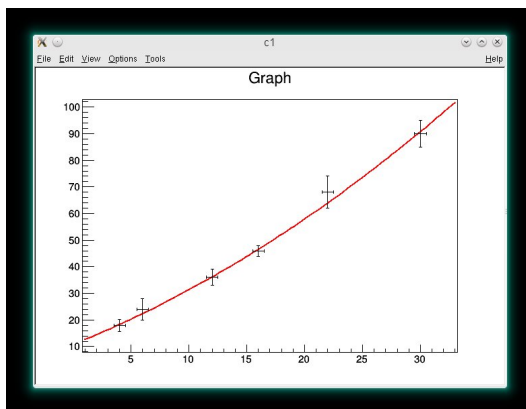
```
root [2] gr->Draw("AP");
```

- 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“

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



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

Arbeitsvorschlag:

- Zum ROOT Setup

ROOT ist im CIP Pool im directory `/janus/software/root/root-6.06.02` zu finden. Setzen sie ROOTSYS und ergänzen sie PATH und LD_LIBRARY_PATH. Dies muss entweder in jeder shell gemacht werden oder im startup File (`.bashrc` oder im CIP Pool in `.profile`). Fügen Sie die Einträge dem `.profile` File hinzu.

- Bestimmen Sie die Summe $\sum_{k=1}^N \frac{x^k}{k}$ für $N=10$ und $x = 3$.

- Stellen Sie die verschiedenen Besselfunktionen aus `::TMath` graphisch dar.
- Benutzen Sie die Klasse `TLorentzVector` um die invariante Masse der Vierervektoren `u(3., 10., 40., 10.)` und `v(1., 0., 10., 3.)` zu bestimmen. Bilden Sie die Operatoren `=`, `+`, `+=`. Bestimmen Sie die transversale Komponente des Summenvektors von `u` und `v`.
- Erzeugen Sie ein Text File mit gedachten Messungen und benutzen Sie die Klasse `TgraphErrors("myFile.txt")` zur Darstellung. Probieren Sie die Fit Funktionalität aus.

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)
```

- 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

myMacro.C:

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


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 ausgeführt
-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



```
# 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 ;
}
```

<https://root.cern.ch/doc/v608/classTRandom3.html>

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

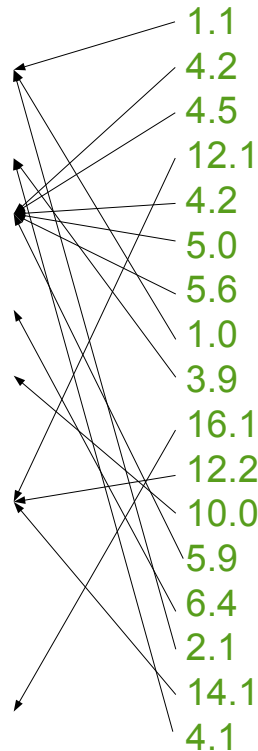
ROOT - Histogramme

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

0 Underflow

1	0-2	3
⋮	2-4	2
⋮	4-6	5
⋮	6-8	1
⋮	8-10	1
⋮	10-12	0
⋮	12-14	3
⋮	14-16	0
9	16-18	1

10 Overflow



- Histogramm Klassen in ROOT:
TH1F, TH1D (1 dimensional)
TH2F, TH2D (2 dimensional)

- 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"
```

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 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.



```
# 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

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);
```

pointer auf das TCanvas Objekt C Canvas Name Titel Pixel Koordinaten von oben links Breite / Höhe des Fensters

```
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);
```

```
h->Scale(1./NumEntries);
```

```
h->Draw();
```

Normieren des Histogramms h auf die Fläche 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 Histogramms in das ROOT file

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

Lesen des Histogramms

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() Erzeugen einer Kopie des Histogramms h  
hsum->SetName("hsum");  
hsum->Add(hbck,1.0); Addieren von hbck zu h mit Skalenfaktor 1
```

```
TH1F *h3 = new TH1F ("h3","h1+h2",nBins,xMin,xMax);  
h3->Add(h1,h2); Addieren von 2 Histogrammen  
h3->Add(h1,h2,1.,-1.); Subtrahieren von 2 Histogrammen
```

- Histogramme h1,h2,h3 haben das gleiche binning!
- Sonst müssen Bingenzen 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);  
}  
h->Draw("COLZ");  
h->Draw("CONTZ");  
h->Draw("LEGO");
```

Gauss Verteilung, Mean=0 und Breite =1

Gauss Verteilung, Mean=1 und Breite =2

Farbänderungen entsprechend der Einträge in z

Contour Plot bezüglich z

Lego Plot Entries in z

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, -1);  
TProfile *profY = h2D->ProfileY("MyProfileY", 1, -1);
```

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 Standardabweichung. 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?

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 dem 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 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 Histogrammeigenschaften 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]);
```

Anpassung des Gauss Models an die Histogramme

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

```
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);}
```

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);  
h→SetTitle("my Title");  
h→SetTitle("my x Axis");  
h→SetAxisRange(10.,25.);  
h→SetMarkerColor(kBlue);  
h→SetMarkerSize(1.);  
h→SetMarkerStyle(20);  
h→SetFillColor(kGreen);  
h→SetFillStyle(3004);  
h→Daw("e");  
h→Print();
```

Linienfarbe auf rot setzen
Titel setzen
Achsenbeschriftung
Bereich ändern
Symbolfarbe auf blau setzen
Symbolgröße setzen
Symbol setzen
Histogrammfarbe
Diagonale Linien unter Histogramm
Zeichnen, Einträge mit Fehler
Info am Bildschirm

Nach Änderungen mit obigen Methoden muss das Histogramm erneut gezeichnet werden oder das Canvas muss einen update erhalten.

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

- 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 Parameter

```
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)

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

Anzahl der Parameter

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();
}
```

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()
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, TRandom2, und TRandom3 (Mersenne-Twister Algorithmus, lange Periode von $2^{19937} - 1$, 45 ns pro call).

- 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
```

Weitere Methoden 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
```


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);
```

Oder

Funktion

Zahl der Einträge

```
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);
```

myFunc.C

```
h->FillRandom("f2",10000);
```

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);
```

Funktions- und
Parameterdefinition

```
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();  
}
```

Zufallszahlen
verteilt wie f1

Gaussverteilte
Zufallszahlen mit
Mittelwert und Sigma

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

Wahrscheinlichkeitsverteilungen

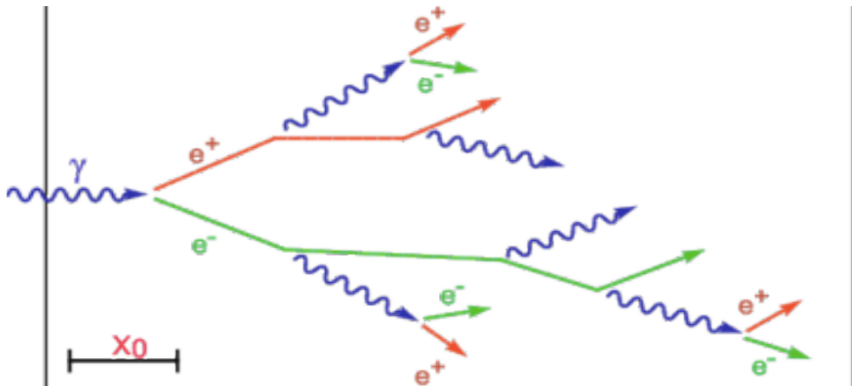
- Referat von Herrn Stezura

pdf

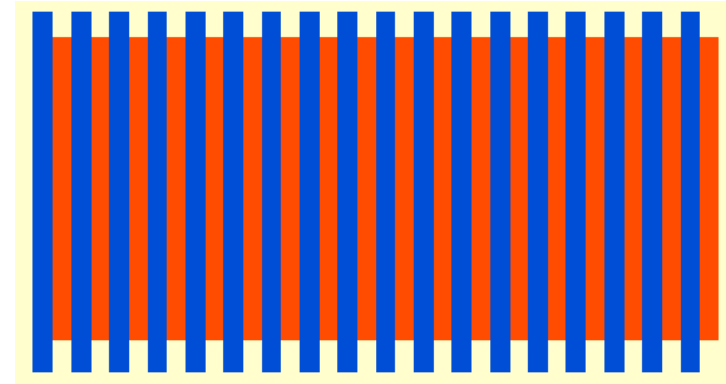
myMaxwell.cc

poisson.cc

Energiemessung von Photonen



Sampling Kalorimeter



Homogene Kalorimeter



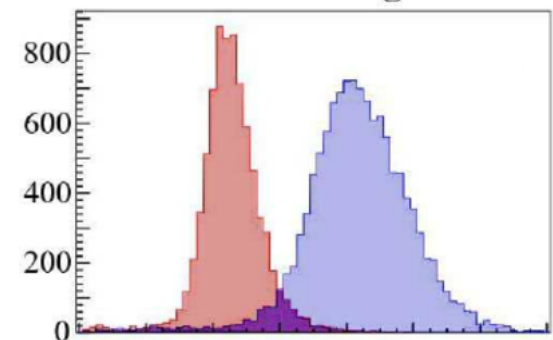
Beiträge zur Energieauflösung:

Sampling / Quantum Fluktuationen $\sigma_s/E \sim \frac{a}{\sqrt{E}}$ $a = 10\%$

Elektronik Noise $\sigma_N/E \sim \frac{b}{E}$ $b = 0.1 \text{ GeV}$

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

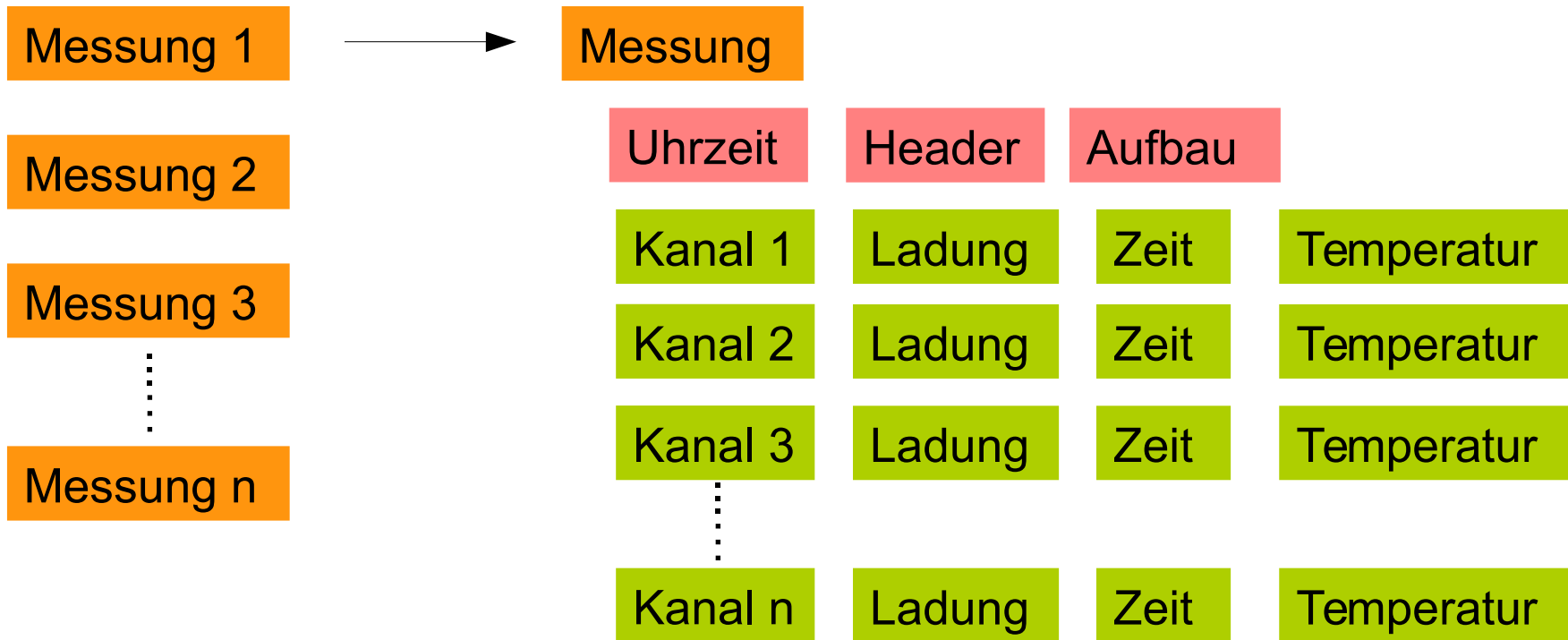
Die Energieauflösung σ/E eines elektromagnetischen Kalorimeters zur Energiemessung von Photonen besteht aus 3 Anteilen mit verschiedenen Energieabhängigkeiten. Die gesamte Energieauflösung ist die quadratische Summe der einzelnen Anteile.

Arbeitsvorschlag:

- Schreiben Sie eine TF1 Funktion für σ und eine für σ/E als C++ Funktion. Zeichnen Sie beide in ein Canvas für 1-150 GeV.
- In das Canvas für σ/E soll der funktionale Verlauf der einzelnen Anteile eingezeichnet werden.
- 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.

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. Spezielle Techniken sorgen für effizienten Input/Output.

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.

```
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 myRooNtuple.root
```

ROOT File öffnen

```
root[0] Attaching file myRooNtuple.root as _file0...
```

```
root[1] TBrowser T
```

Inhalt ansehen

```
root[2] _file0->ls()
```

```
... .
```

```
root[3] TCanvas *myCanvas = new Tcanvas()
```

Neues Canvas erzeugen

```
root[4] ntuple->Draw("px:E")
```

```
root[5] ntuple->Draw("px:E", "p>600")
```

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");
```

Specify TTree name

```
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();
```

```
}
```

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");
```

Specify TTree name

```
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();
```

```
}
```

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");
```

Specify TTree name

```
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++) {
```

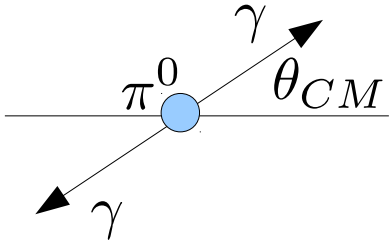
```
    t->GetEntry(i);
```

```
    cout << "Event " << nEvent << "    p[0] = " << p[0] << endl;
```

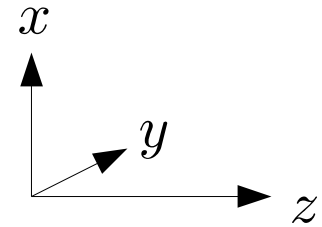
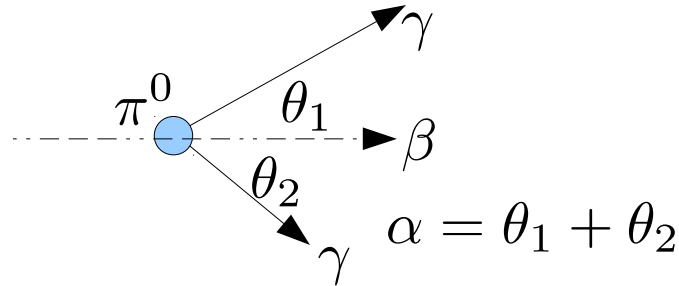
```
}
```

Simulation des π^0 Zerfalls

Schwerpunktsystem:



Laborsystem:



Relativistische Variablen:

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

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.

Simulation des π^0 Zerfalls

Die maximale und minimale Energie im Laborsystem ist:

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

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

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

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, dass 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 Viervektoren 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.
- Ergänzen Sie ihr Programm in dem Sie die Messung des generierten Zerfalls simulieren. Dazu werden die generierten Werte der Photonenergie und der Photonenwinkel 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.

`pi0DecaySim.C`

`pi0DecayCalo.C`

Datenanpassung

- Fitten – das wichtigste Tool der Datenanalyse

Mathematische Prozedur ein Modell and Datenpunkte anzupassen. Das Modell wird durch eine Funktion beschrieben, die freie Parameter besitzt. Die Parameter werden beim Anpassen so verändert, dass die folgende Grösse minimal wird:

$$\chi^2 = \sum \left(\frac{y_i - f(x_i)}{\sigma_i} \right)^2$$

Die Funktion wird als χ^2 Funktion bezeichnet. Ein guter Fit hat $\chi^2 / ndf \approx 1$, wobei ndf die Anzahl der Freiheitsgrade ist.

Voraussetzung für eine „erfolgreiche“ Datenanpassung sind

- der Verlauf der Funktion kann grundsätzlich die Daten beschreiben
- die Startparameter sind so gesetzt, das die Funktion ungefähr die Daten beschreibt

- Fitten – in ROOT mit Hilfe von speziellen Programm Paketen

- Minuit wird in ROOT zum fitten verwendet
- RooFit
- RooStats

ROOT - Histogramme

Erweiterungen zu
myHisto.C

- Histogramme fitten

```
TH1F *h = new TH1F ("myhist", "Altersverteilung", 60, 0., 60.);  
...  
h->Fill(age);  
...  
h->Fit("gaus", "M", "", 15., 25);
```

Fit Funktion

Fit Bereich

Arbeitsvorschlag:

Probieren Sie die diskutierten Methoden aus und ändern Sie dazu myHisto.C

myHistoExtended.C

- Nützliche Beispiele

```
TAxis *axis = h->GetXaxis();  
Find a Bin bin_number = axis->FindBin(x_value);
```

```
TH1F * h3 = new TH1F ("h3", "h1/h2", nBin, xMin, xMax);  
Divide 2 Histos h3->Divide(h1, h2);
```

ROOT – Darstellung von Messungen

- Example code zur Darstellung von Messungen

Erzeugen des Graphen

Daten in Form von Arrays

```
TGraphErrors *g = new TGraphErrors(nP,X,Y,ErrorX,ErrorY);
```

Canvas um den Graphen darzustellen

```
TCanvas *myC = new TCanvas ("myC", "My Histogram Results",  
0,0,800,600);
```

Funktion für das Datenmodell

Position und Grösse

```
TF1 * f = new TF1("f", "[0]+[1]*x*x", 0., 7.0);
```

Anpassung der Funktion an den Graphen

```
g->Fit("f");
```

Definition der Beschreibung

Position und Grösse

```
TLegend *leg = new TLegend(0.1,0.7,0.3,0.9,"Time vs Position");
```

Erzeugen eines Output Files

```
myC->Print("myGraph.pdf");
```

Endung legt Filetyp fest

Vollständiges Beispiel in myGraph.C

```
>$ root myGraph.C
```

```
>$ g++ myGraph.C `root-config --cflags --glibs`
```

Erzeugt kein Canvas!

ROOT – Python Interface

ROOT hat ein Python binding (PyROOT) und vereinigt damit compilierte C++ Bibliotheken mit der Flexibilität von Python.

- Ein Beispiel : Füllen eines Histogramms mit einer Polynomial 1.Grades Zufallsverteilung.

```
>$ python
Python 2.7.8 (default, Sep 30 2014, 15:34:38) [GCC] on linux2
Type "help", "copyright", "credits" or "license" for more
information.
>>> import ROOT
>>> h=ROOT.TH1F("myPyHisto", "Productivity;Python Knowledge;
Productivity", 64, 0, 63)
>>> h.FillRandom("poll")
>>> h.Draw()
```

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