Introduction to ROOT

Philip Bechtle

DESY

February 2011

Inhalt

Einführung in das Datenanalysepaket ROOT
(kostenlos erhältlich unter http://root.cern.ch):

- Starten und Beenden des Programmes
- Erzeugen von Funktionen und Histogrammen
- Grafische Darstellung von Funktionen und Histogrammen
- Benutzung von Makros
- (X)Emacs: Ein Editor (andere sagen, ein Betriebssystem . . .)
- C++-Intermezzo
- Verwendung des Zufallszahlengenerators

Ausführliche Einführung erhältlich unter:
http://root.cern.ch/root/doc/RootDoc.html

Vollständige Dokumentation aller Befehle:
http://root.cern.ch/root/Reference.html

Was dieser Vortrag nicht leisten kann . . .

ROOT verwendet C++-Syntax. Eine umfassende Einführung in diese Programmiersprache ist im Rahmen dieser Übungen nicht möglich. Hier wird nur das wichtigste C++-Wissen vermittelt, was für den ersten ROOT-Umgang nötig ist.
Mehr gibt es nach Bedarf in den folgenden Übungen . . .

Was macht ROOT?

- Darstellung von Funktionen
- Darstellung von Daten (Histogramme, Scatter-Plots, etc . . .)
- Fit von Modellen (Funktionen) an Daten (χ^2, ML, etc . . .)
- Verwalten von Unsicherheiten (Statistische Fehler, Fehlerfortpflanzung bei Operationen mit Histogrammen, etc . . .)
- Verwalten von großen Datenmengen (ntuple, TTree, etc . . .)
- Entwicklung von Selektionen (TMVA, etc . . .)
- Wenn man möchte: graphische Oberflächen mit Darstellung der Daten . . .
- . . .

Und los gehts

- Starten von ROOT:

```
bechtle@momo:~/Lectures/Statistik08/Uebung/LoesungenProgramme > root
  ****************************************
  *                                      *
  *        W E L C O M E  to  R O O T    *
  *                                      *
  *   Version   5.16/00      27 June 2007  *
  *                                      *
  *  You are welcome to visit our Web site  *
  *          http://root.cern.ch         *
  *                                      *
  ****************************************

Compiled on 25 November 2007 for linuxx8664gcc with thread support.

CINT/ROOT C/C++ Interpreter version 5.16.21, June 22, 2007
Type ? for help. Commands must be C++ statements.
Enclose multiple statements between { }.
root [0]
```

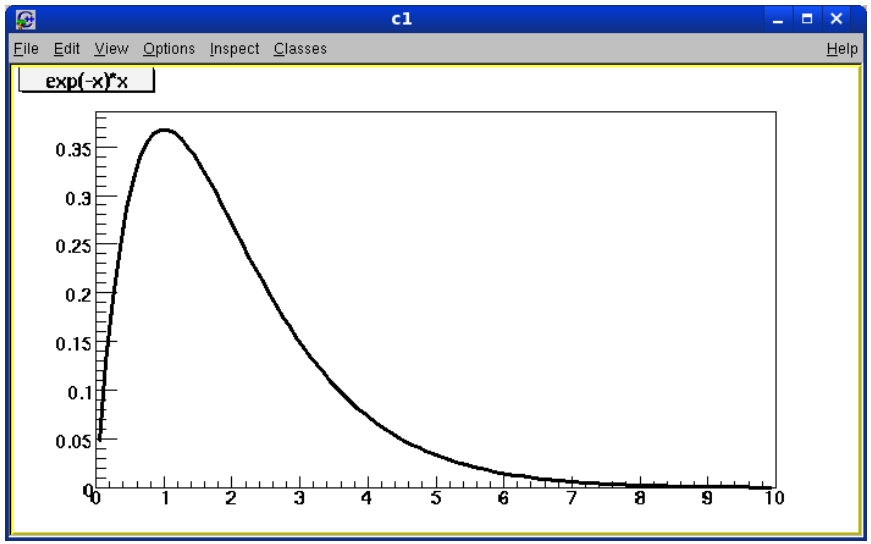
- Beenden von ROOT:

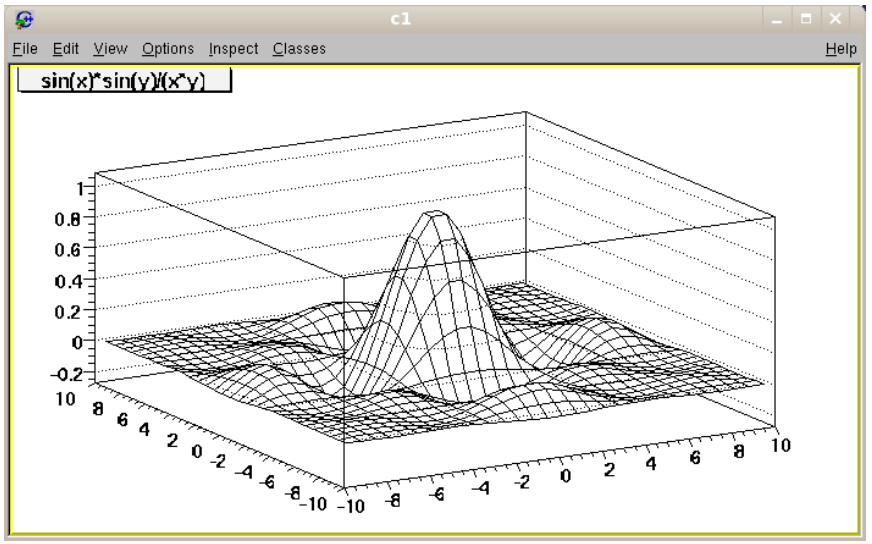
```
root [0] .q
bechtle@momo:~/Lectures/Statistik08/Uebung/LoesungenProgramme >
```

Funktionen und ihre Darstellung

- Eindimensionale Funktionen:

```
root [0] TF1 f1("func1","exp(-x)*x",0,10)
root [1] f1.Draw()
```





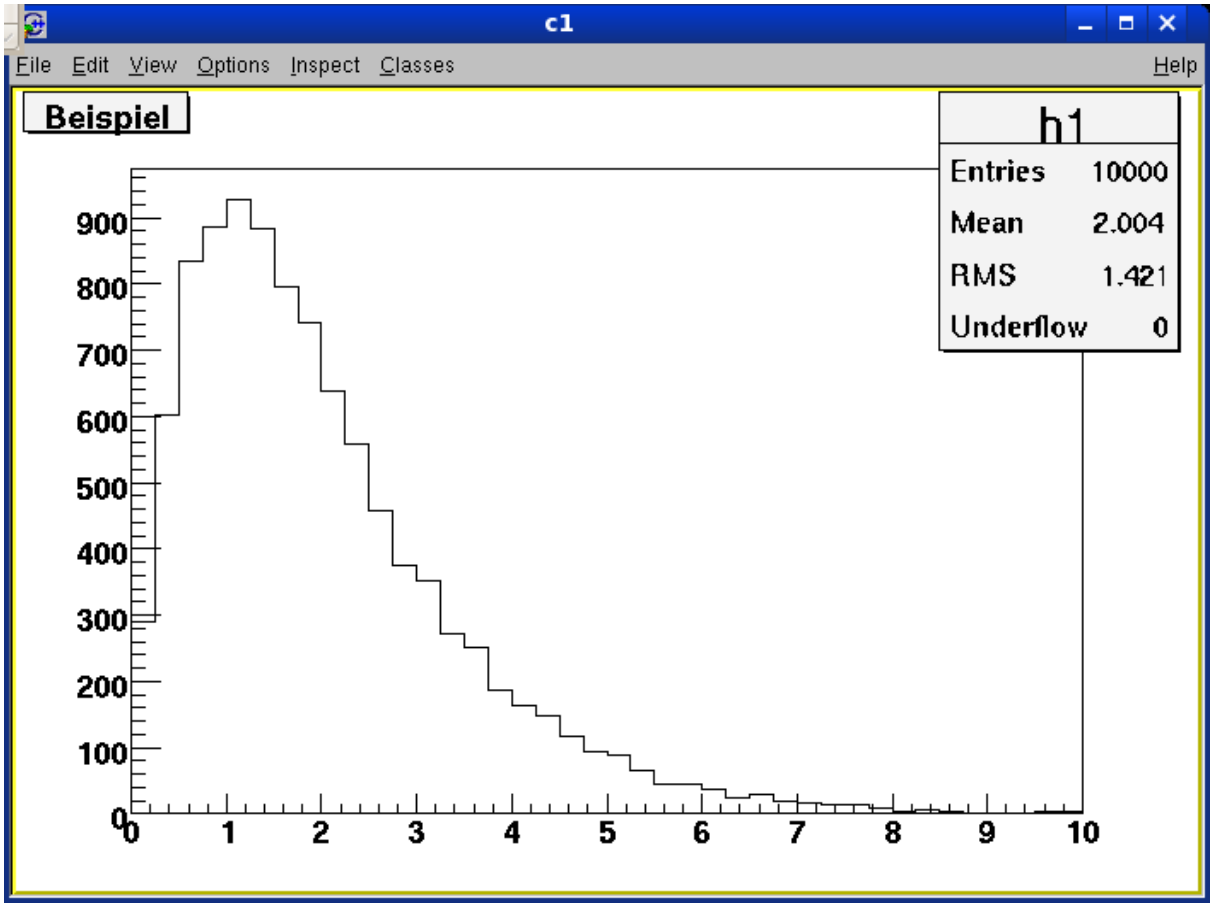
- Analog: Zweidimensionale Funktionen:

```
root [2] TF2 f2("func2","sin(x)*sin(y)/(x*y)",-10,10,-10,10)
root [3] f2.Draw("surf")
```

Das Attribut `surf` ist eine Darstellungsoption

Histogramme und ihre Darstellung

```
root [0] TH1F h1("h1","Beispiel",40,0,10)
root [1] h1.Draw() // ergibt ein leeres Histogramm
root [2] TF1 f1("func1","exp(-x)*x",0,10)
root [3] h1.FillRandom("func1", 10000)
root [4] h1.Draw() // zeigt das gefuellte Histogramm
```



Definition zwei- und dreidimensionaler Histogramme analog

Makros

- (Neben vielen anderen) Einer der größten Vorteile von ROOT: Befehle können in Macros oder ganzen Programmen gespeichert und beliebig kompiliert, wiedervewendet oder neu verwendet werden.
- Sehr einfaches Beispiel: Schreibe eine Datei mit dem Imhalt:

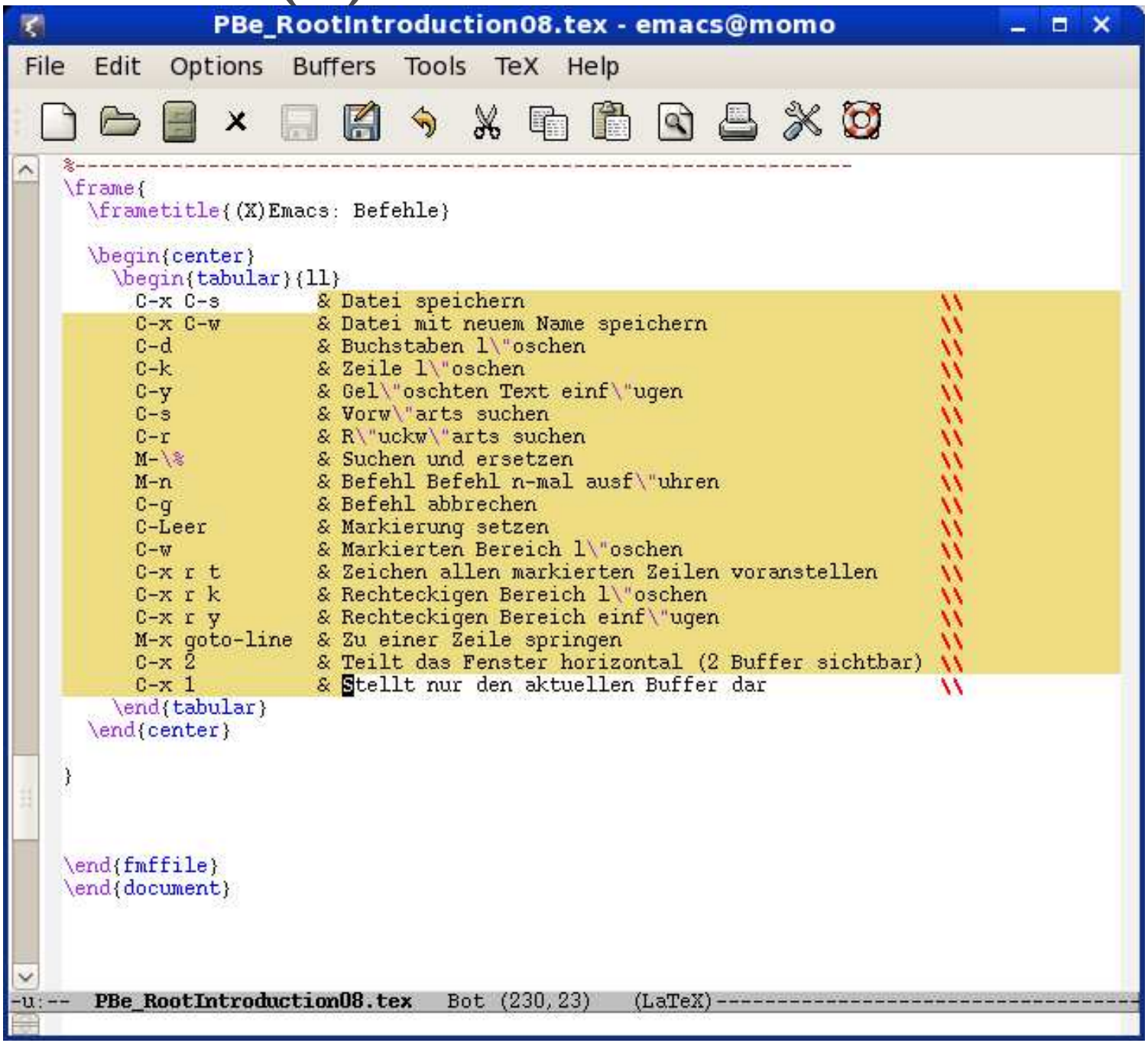
```
void macro1 () {
  TH1F h1("h1","Beispiel",40,0,10);
  TF1 f1("func1","exp(-x)*x",0,10);
  h1.FillRandom("func1", 10000);
  h1.DrawCopy();
}
```

- Ausführung in ROOT:

```
root [0] .L macro1.cpp
root [1] macro1();
```

- Benutze einen guten Editor zum Verwalten des Codes. Z.B. (X)Emacs

(X)Emacs: Ein Editor



(X)Emacs: Befehle

C-x C-s	Datei speichern
C-x C-w	Datei mit neuem Name speichern
C-d	Buchstaben löschen
C-k	Zeile löschen
C-y	Gelöschten Text einfügen
C-s	Vorwärts suchen
C-r	Rückwärts suchen
M-%	Suchen und ersetzen
M-n	Befehl Befehl n-mal ausführen
C-g	Befehl abbrechen
C-Leer	Markierung setzen
C-w	Markierten Bereich löschen
C-x r t	Zeichen allen markierten Zeilen voranstellen
C-x r k	Rechteckigen Bereich löschen
C-x r y	Rechteckigen Bereich einfügen
M-x goto-line	Zu einer Zeile springen
C-x 2	Teilt das Fenster horizontal (2 Buffer sichtbar)
C-x 1	Stellt nur den aktuellen Buffer dar

Komplexere Funktionen

- Definiere Funktion:

```
Double_t chi2Function(Double_t *x, Double_t *par)
{
    Double_t retval = 0;
    if ( TMath::Power(2, 0.5*par[1]) * TMath::Gamma(0.5*par[1]) ) {
      retval = (par[0]*(TMath::Power(x[0], 0.5*par[1] - 1)
        * TMath::Exp(-0.5*x[0]) /
        (TMath::Power(2, 0.5*par[1]) * TMath::Gamma(0.5*par[1]))));
    }
    return retval;
}
```

- Benutze sie:

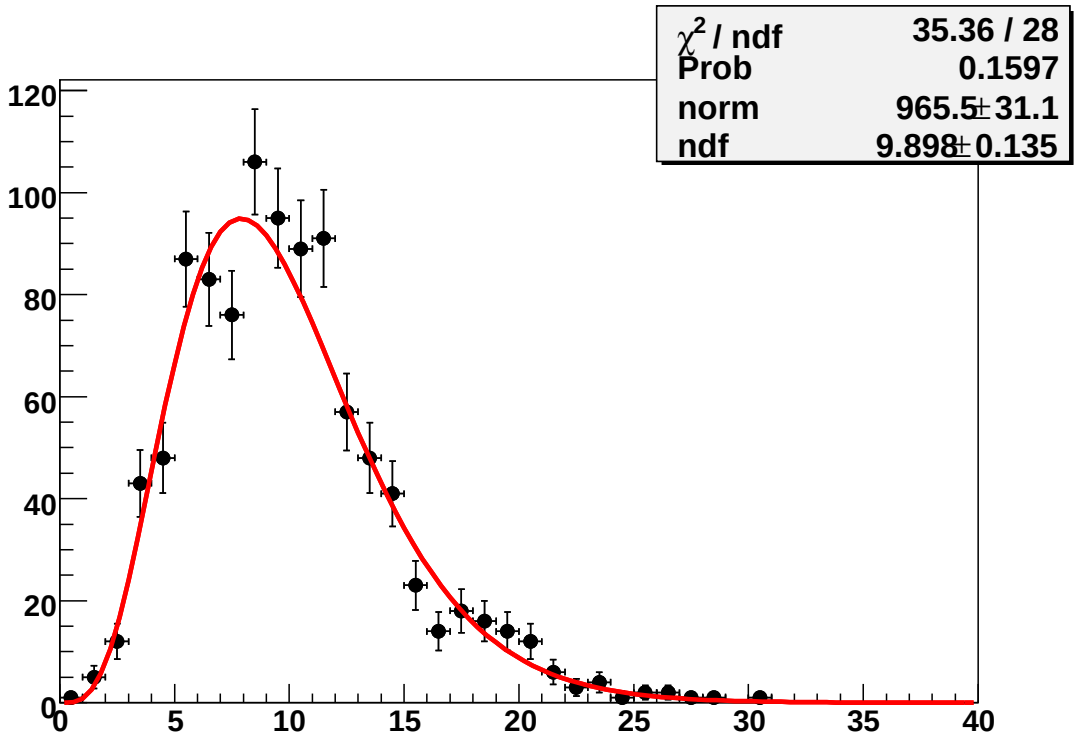
```
void macro3 () {
  chi2func = new TF1("chi2func", chi2Function, 0., 30, 2 );
  chi2func->SetParNames("norm", "ndf");
  chi2func->SetParameter(0, 100. );
  chi2func->SetParameter(1, 10.);
  chi2func->Draw();
}
```

Fits

```
void macro2 () {
  const int nEvents   = 1000;
  const int ndf = 10;
  TH1D* chis2dist = new TH1D("chi2dist","",4*ndf,0.,(double)4*ndf);
  TRandom3* randomGen = new TRandom3();
  for (int iEvt = 0; iEvt < nEvents; iEvt++) {
    double y = 0.;
    for (int nObs = 0; nObs < ndf; nObs++) {
      double x = randomGen->Gaus(0.,1.);
      y += x*x;
    }
    chi2dist->Fill(y);
  }
  TCanvas* c = new TCanvas();
  gStyle->SetOptStat(0);
  gStyle->SetOptFit(111111);
  chi2func = new TF1("chi2func", chi2Function, 0., 4.*ndf, 2 );
  chi2func->SetParNames("norm", "ndf");
  chi2func->SetParameter(0, 0.5*(double)nEvents);
  chi2func->SetParameter(1, ndf);
  chi2func->SetLineColor(kRed);
  chi2dist->Fit(chi2func,"E");
  c->Print("chi2Dist.eps");
  return;
}
```

Resultat

```
FCN=35.3557 FROM MINOS     STATUS=SUCCESSFUL     40 CALLS          87 TOTAL
                     EDM=3.69603e-09    STRATEGY= 1      ERROR MATRIX ACCURATE
EXT PARAMETER                          PARABOLIC         MINOS ERRORS
NO.   NAME      VALUE            ERROR      NEGATIVE      POSITIVE
 1  norm          9.65527e+02   3.10872e+01  -3.10882e+01   3.10862e+01
 2  ndf           9.89801e+00   1.34986e-01  -1.34149e-01   1.35912e-01
```



Resultat: Korrelationen

- Wie wir schon gesehen haben, sind korrelationen (zwischen Obserablen oder Parametern) extrem wichtig! Leider gehören sie nicht zur Standard-Ausgabe . . .

```
...
chi2dist->Fit(chi2func,"E","E1");
TVirtualFitter *fitterObject = TVirtualFitter::GetFitter();
TMatrixD *covMatrix = new TMatrixD(2,2,fitterObject->GetCovarianceMatrix());
covMatrix->Print();
```

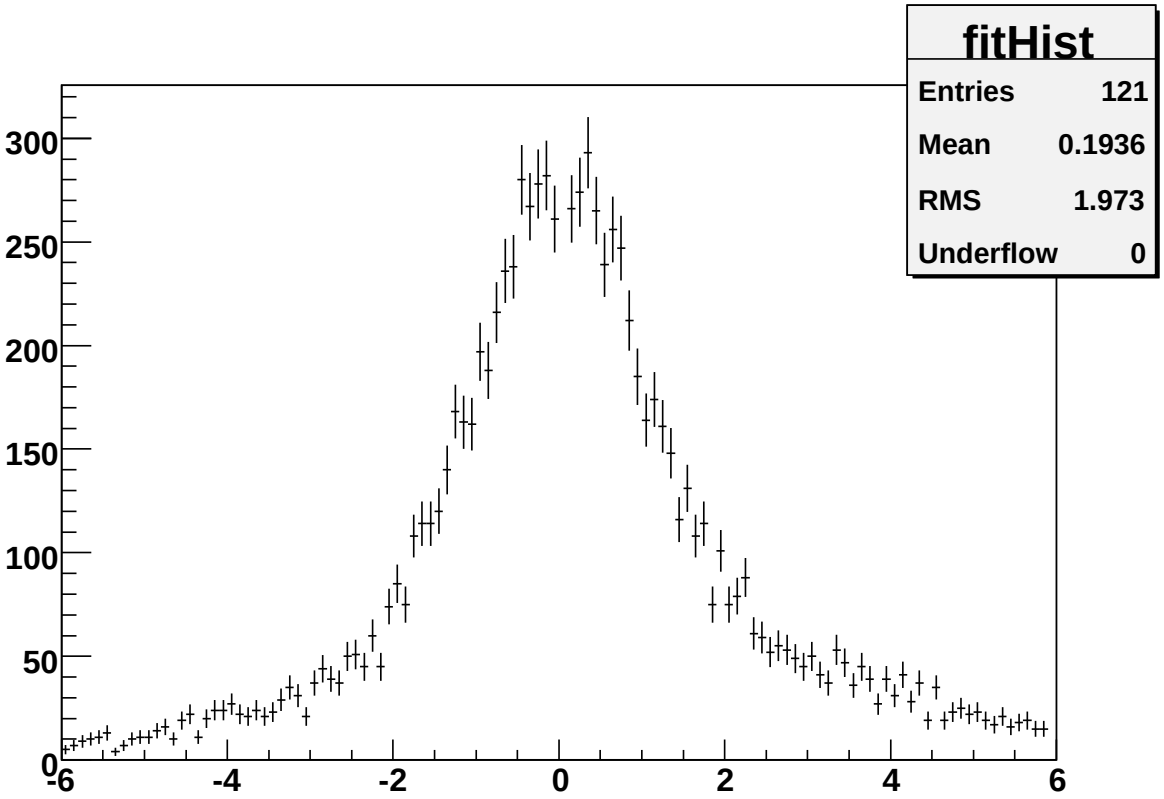
- Dann sieht das Ergebnis so aus:

```
2x2 covMatrix is as follows

     |      0    |      1    |
-------------------------------
   0 |      966.4    0.009958
   1 |   0.009958     0.01822
```

Histogramme und das Einlesen von Daten

```
void macro5 () {
  ifstream in;
  in.open("something.dat");

  TH1D* fitHist = new TH1D("fitHist","",120,-6.,6.);

  int nlines=0;
  while (!in.eof()) {
    nlines++;
    double massDiff = 0;
    int nEvents = 0;
    in >> massDiff >> nEvents;
    // if (in.eof()) break;
    cout << nlines << " " << fitHist->FindBin(massDiff)
         << " " << massDiff << " " << nEvents << " " << sqrt(nEvents) << endl;
    fitHist->SetBinContent(fitHist->FindBin(massDiff),nEvents);
    fitHist->SetBinError(fitHist->FindBin(massDiff),sqrt(nEvents));
  }
  fitHist->DrawCopy();

  in.close();
}
```

Ergebnis


fitHist
Entries 121
Mean 0.1936
RMS 1.973
Underflow 0
300
250
200
150
100
50
0
-6
-4
-2
0
2
4
6

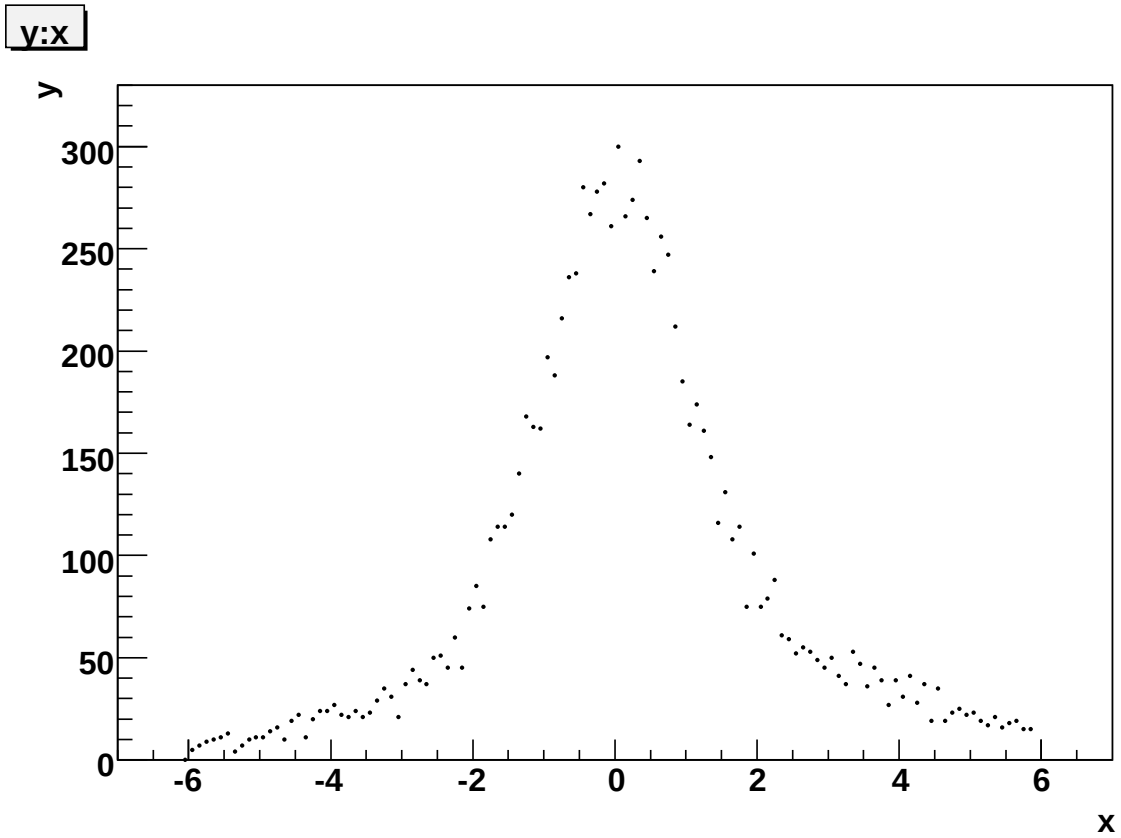
Ntuple und das Einlesen von Daten

```
#include "Riostream.h"
void macro6 () {
   ifstream in;
   in.open("something.dat");

   double x,y;
   Int_t nlines = 0;
   TFile* file = new TFile("something.root","RECREATE");
   TNtuple *ntuple = new TNtuple("ntuple","data from ascii file","x:y:erry");

   while (1) {
      in >> x >> y;
      if (!in.good()) break;
      cout << x << " " << y << endl;
      ntuple->Fill(x,y,sqrt(y));
      nlines++;
   }

   printf(" found %d pointsn",nlines);
   in.close();
   file->Write();

   return;
}
```

Interaktive Verwendung des NTuples

```
root [0] TFile f("something.root")
root [1] f.ls();
TFile**         something.root
 TFile*         something.root
  KEY: TNtuple  ntuple;1        data from ascii file
root [2] f->Print();
TFile: name=something.root, title=, option=READ
root [3] ntuple->Print();
******************************************************************************
*Tree    :ntuple    : data from ascii file                                     *
*Entries :      120 : Total =            3602 bytes  File  Size =       1479 *
*        :          : Tree compression factor =   1.00                       *
******************************************************************************
*Br    0 :x         :                                                         *
*Entries :      120 : Total  Size=       1084 bytes  One basket in memory    *
*Baskets :        0 : Basket Size=      32000 bytes  Compression=   1.00     *
*.............................................................................*
*Br    1 :y         :                                                         *
*Entries :      120 : Total  Size=       1084 bytes  One basket in memory    *
*Baskets :        0 : Basket Size=      32000 bytes  Compression=   1.00     *
*.............................................................................*
*Br    2 :erry      :                                                         *
*Entries :      120 : Total  Size=       1102 bytes  One basket in memory    *
*Baskets :        0 : Basket Size=      32000 bytes  Compression=   1.00     *
*.............................................................................*
root [4] ntuple->Draw("y:x");
```

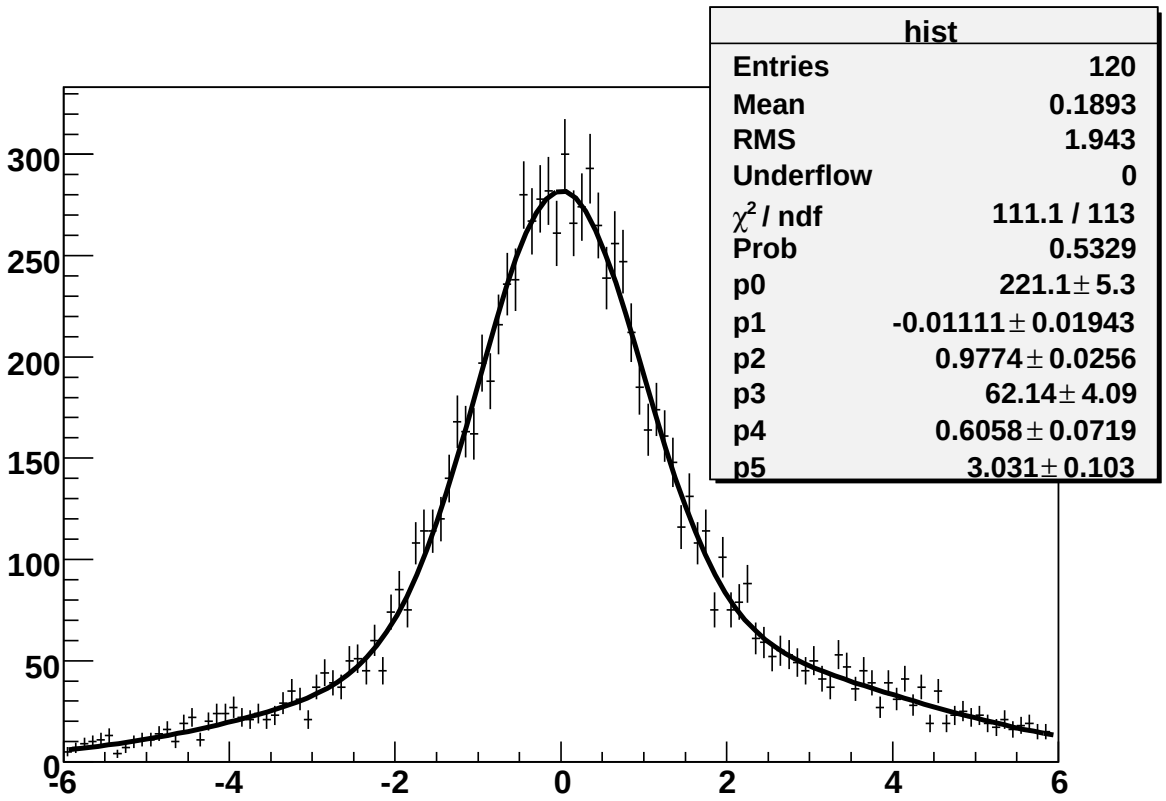
Ergebnis



Verwendung des NTuples

```
void macro7 () {
TFile* f   = new TFile("something.root","READ");
TH1D* hist = new TH1D("hist","",120,-6.,6.);
TNtuple* ntuple = f->Get("ntuple");
float x, y, erry;
ntuple->SetBranchAddress("x",    &x   );
ntuple->SetBranchAddress("y",    &y   );
ntuple->SetBranchAddress("erry", &erry);
if (ntuple==0) return;
int nEntries = ntuple->GetEntries();
for (int i = 0; i < nEntries; i++) {
  ntuple->GetEntry(i);
  cout << x << endl;
  hist->SetBinContent(hist->FindBin(x),y   );
  hist->SetBinError  (hist->FindBin(x),erry);
}
TF1*  gaussFunc = new TF1("gaussFunc","gaus(0)+gaus(3)",-6.,6.);
gaussFunc->SetParameter(0, 100. );
gaussFunc->SetParameter(1, 0.);
gaussFunc->SetParameter(2, 1.);
gaussFunc->SetParameter(3, 30. );
gaussFunc->SetParameter(4, 0.5);
gaussFunc->SetParameter(5, 3.);
hist->Fit("gaussFunc");
return;
```

Ergebnis


hist
Entries 120
Mean 0.1893
RMS 1.943
Underflow 0
χ² / ndf 111.1 / 113
Prob 0.5329
p0 221.1 ± 5.3
p1 -0.01111 ± 0.01943
p2 0.9774 ± 0.0256
p3 62.14 ± 4.09
p4 0.6058 ± 0.0719
p5 3.031 ± 0.103
300
250
200
150
100
50
0
-6
-4
-2
0
2
4
6

Ergebnis: Korrelationen

Covarianzmatrix:

	0	1	2	3	4	5
0	27.69	0.007424	0.01044	-11.68	0.1262	0.2874
1	0.007424	0.0003775	5.578e-05	-0.01204	-0.0002186	0.0003122
2	0.01044	5.578e-05	0.0006569	-0.07874	0.0008346	0.001658
3	-11.68	-0.01204	-0.07874	16.71	-0.1669	-0.3855
4	0.1262	-0.0002186	0.0008346	-0.1669	0.005173	0.004246
5	0.2874	0.0003122	0.001658	-0.3855	0.004246	0.01068

Correlationsmatrix:

	0	1	2	3	4	5
0	1	0.07261	0.07742	-0.5428	0.3334	0.5284
1	0.07261	1	0.112	-0.1516	-0.1564	0.1555
2	0.07742	0.112	1	-0.7516	0.4527	0.6259
3	-0.5428	-0.1516	-0.7516	1	-0.5678	-0.9126
4	0.3334	-0.1564	0.4527	-0.5678	1	0.5713
5	0.5284	0.1555	0.6259	-0.9126	0.5713	1

Verwenden des Zufallszahlengenerators

- Root stellt mehrere Zufallszahlengeneratoren zur Verfügung.
- Globales Objekt gRandom ist interaktiv immer Verfügbar:

```
TH1F histo("histo","Gaussverteilung",20,-5,5);
for (int i=0; i<10000; i++) {
  histo.Fill(gRandom.Gaus(0,1),1);
}
histo.Draw();
```

- Alternativ: Erzeuge Objekt selbst:

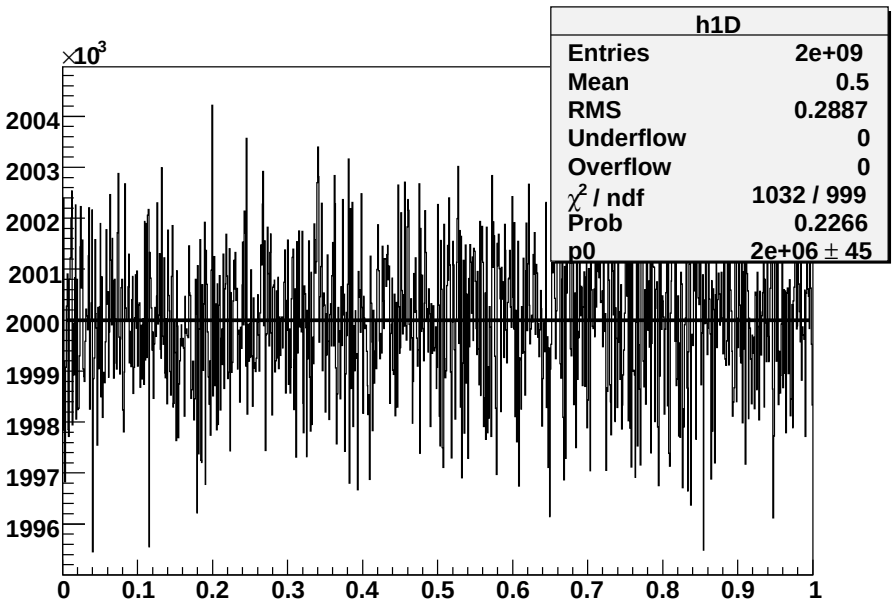
```
TRandom1 random1;
TRandom2 random2;
TRandom3 random3;
```

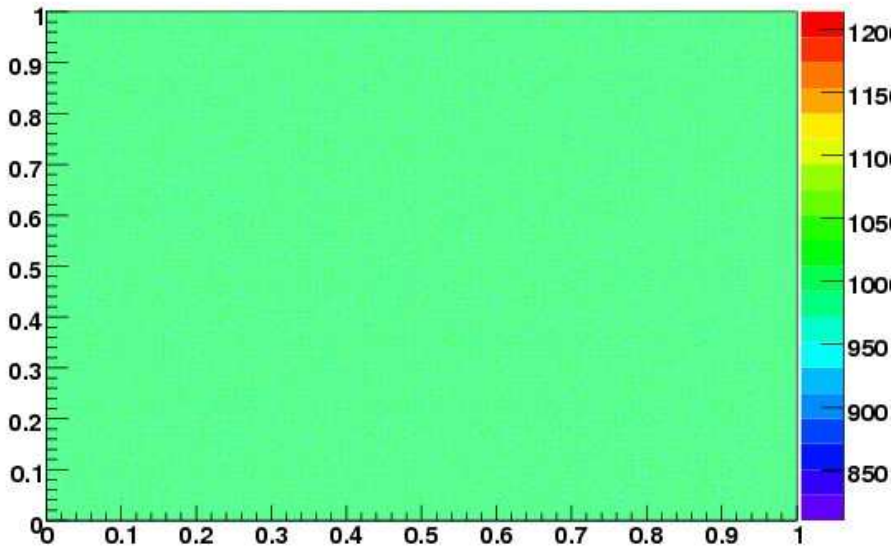
- TRandom3 hat längste Periode und beste Zufälligkeit (geringe Korrelationen zwischen konsekutiven Zufallszahlen)
- weitere eingebaute PDFs:

```
random3.Rndm()
random3.Exp(tau)
random3.Uniform(min, max)
```

Test des Zufallszahlengenerators

- Einfache Tests:
- Lange Periode
- Keine Korrelationen zwischen konsekutiven Zufallszahlen
- Fit an flache Zufallsverteilung liefert gute Prob(χ^2)





Übung

Bestimmen Sie das Integral der Funktion

$$f(x) = x^2$$

im Intervall von 0 bis 1 mittels Monte-Carlo-Integration! Würfeln Sie dazu Zufallszahlen, die zufällig über ein Quadrat der Größe 1×1 verteilt sind, und zählen Sie die Anzahl der gewürfelten Punkte in diesem Quadrat und diejenige unterhalb der Funktionskurve! Würfeln Sie 100 mal, und tragen Sie den Näherungswert für den i-ten Iterationsschritt in das i-te Bin eines Histogramms ein!