

ROOT-Einführung im Rahmen des FPs

Björn Penning



Zum Tutorial

- Dieses behandelt elementare Techniken im Umgang von ROOT zur Auswertung und Visualisierung von Versuchen im Rahmen des FP
- Das Tutorial ist so ausgelegt, dass die wichtigsten für das Praktikum notwendigen Beispiele angesprochen werden und das Tutorial ebenfalls als Basis-Referenz dient.
- Dies ist kein C/C++ Kurs. ROOT verwendet C++ Syntax, aufgrund der Verwendung des in ROOT eingebauten Compiler/Interpreter kann aber sehr einfach programmiert werden
- Die notwendigen Programmier-Kenntnisse sollten sogar für absolute Anfänger in wenigem Minuten erlernbar sein... wenn das nicht der Fall ist wünsche ich schon mal viel Glück in der Diplomarbeit ;-)
- Ein Wort in eigener Sache... ich hätte mit zu Zeiten des FPs ROOT gewünscht. Auf den ersten Blick sieht es schwerer aus als "klicki-bunti-Origin" aber es läuft, ist stabil, einfach und im Prinzip programmiert man alles ein einzige mal, anschließend nur noch Copy-und-Paste. Bei ORIGIN klickt man sich dafür 6 Wochen lang tot....

Was ist ROOT

- ROOT ist ein objektorientiertes Softwarepaket zur Datenanalyse und -visualisierung. Es basiert auf der Programmiersprache C/C++.
- Auf <http://root.cern.ch> finden sich
 - Binaries für versch. Betriebssysteme verfügbar (Linux, Windows, MacOS) und ebenfalls der Quellcode
 - [User's Guide](#) (nicht unbedingt hunderte Seiten ausdrucken, so hilfreich ist er auch nicht)
 - [Reference Guide](#) (sehr hilfreiche Befehlsreferenz)
 - Tutorials, Howto's etc
- Diese Einführung, Quellcode und weitere Infos sind online verfügbar unter

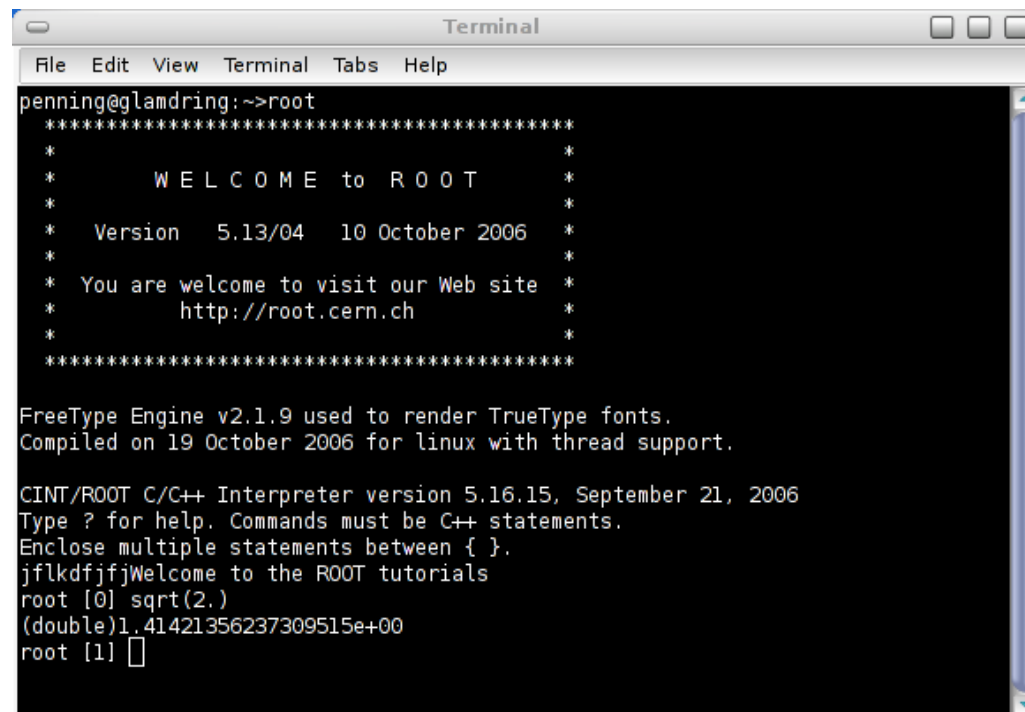
<http://james.physik.uni-freiburg.de/~penning/teaching/root/>

- Bei Fragen stehe ich Euch jederzeit gerne zur Verfügung (3. Stock, Zimmer 03036)



Erste Schritte in ROOT

- in der Kommandozeile auf den CIP-Pool Rechner einfach `root` eingeben
 - man erhält das ROOT-Prompt, einem bei ROOT eingebautem interaktivem C/C++ Interpreter
 - “.q” zum verlassen
- in diese Kommandozeile läßt sich interaktiv Code eingeben, z.B. die Wurzel aus 2



```
Terminal
File Edit View Terminal Tabs Help
penning@glamdring:~>root
*****
*                                     *
*      W E L C O M E  t o  R O O T    *
*                                     *
*   Version   5.13/04   10 October 2006 *
*                                     *
* You are welcome to visit our Web site *
*      http://root.cern.ch              *
*                                     *
*****

FreeType Engine v2.1.9 used to render TrueType fonts.
Compiled on 19 October 2006 for linux with thread support.

CINT/ROOT C/C++ Interpreter version 5.16.15, September 21, 2006
Type ? for help. Commands must be C++ statements.
Enclose multiple statements between { }.
jflkdfjfjWelcome to the ROOT tutorials
root [0] sqrt(2.)
(double)1.41421356237309515e+00
root [1] □
```

Macros in Root

- Im Rahmen dieser Einführung und des Praktikums werden wir versuchen, ROOT nicht interaktiv zu verwenden sondern den Code in eine Datei zu schreiben und diese als Macro auszuführen
 - Macros lassen sich mit jedem beliebigem Editor erstellen (z.B. emacs)
 - Macros lassen sich mit `'.x macro.C'` ausführen oder root starten mit `'#~>root macro.C' (-1 um den Splash Screen loszuwerden)`
- Beispiel:

```
//beispiel.C
{
  gRoot->Reset();  ← Reset des Interpreters
  cout << "Beispiel 1: " << endl;
  cout << "Wurzel aus 2 = " << sqrt(2.) << endl;
}
```

Histogramme

- Ein Histogramm `myH1` mit 10 Bins und einem Wertebereich von 0. bis 1. wird erzeugt durch:

```
TH1F* myH1=new TH1F("myHisto","Distribution 0. to 1.",10,0.,1.);
```

Name

Titel

Anzahl Bins und Bereich

- zum Füllen des Histogramms steht die `Fill()` Methode zur Verfügung, dadurch wird der Wert `x` in das Histogramm eingetragen:

```
myH1->Fill(x);
```

Mittels der `Draw()`-Methode lässt sich das Histogramm darstellen:

```
myH1->Draw();
```

Histogramme

- Das folgende Macro erzeugt das besprochene Histogramm mit drei Werten gefüllt und zeigt dieses an.

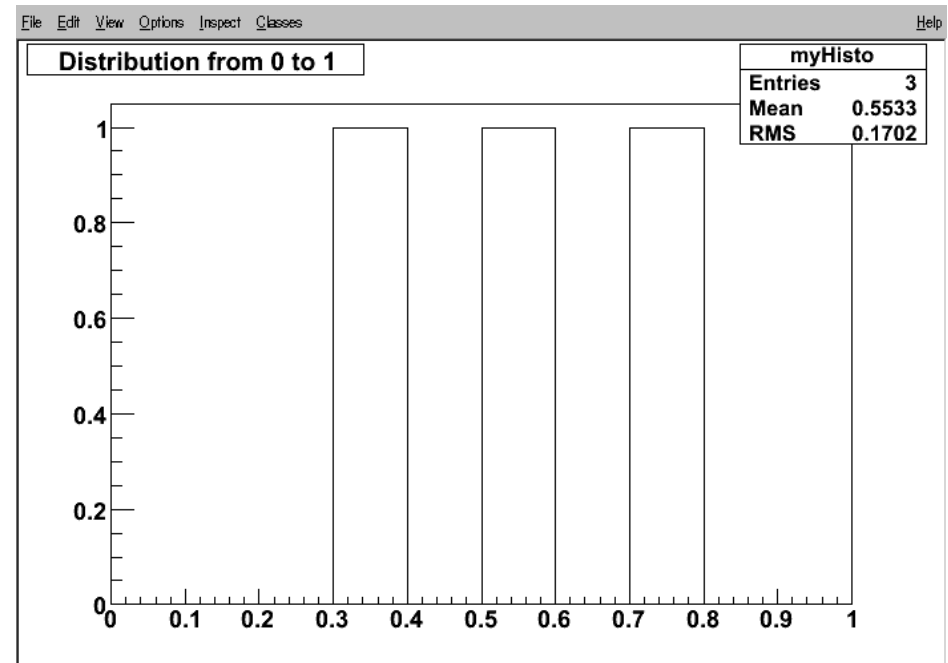
```
//histogram.C
{
  gROOT->Reset();
  gROOT->SetStyle("Plain");
  TH1F* myH1 = new TH1F("myHisto","Distribution from 0 to 1",10,0.,1.);
  myH1->Fill(0.37);
  myH1->Fill(0.78);
  myH1->Fill(0.51);
  myH1->Draw();
}
```

Graph. Darstellung aufräumen

- Die Standard-Darstellungen von Root sind nicht sehr gelungen, Hintergründe z.B. sind leicht schattiert. Folgender Befehl hilft:

```
gROOT->SetStyle("Plain");
```

- Das Ergebnis:



Optionen zum Zeichnen von Histogrammen

- Die `Draw()` Funktion besitzt viele Optionen zum Zeichnen von Histogrammen, einige werden hier angesprochen. Mehr finden sich im ROOT-Manual
 - `"E"` zeichnet Fehlerbalken des Histogramms ein
 - `"SAME"` zeichnet ein Histogramm über ein bereits dargestelltes
 - `"C"` verbindet die Punkte mit einer glatten Kurve
- die meisten Optionen lassen sich kombinieren, z.B. `"SAME, E"`.
- viele Optionen gelten auch für andere Klassen, z.B. `"SAME"` ist ebenfalls für Funktionene `TF1` gültig
- weitere häufig verwendete Funktionen für Histogramme
 - `myH1->SetLineColor(4);`
 - `myH1->SetLineWidth(3);`
 - `myH1->SetFillColor(2);`
 - `myH1->SetFillStyle(3005);`
 - `myH1->SetLineStyle(2);`

Optionen zum Zeichnen von Histogrammen

- zudem können verschiedene Marker für die Datenpunkte verwendet werden, besonders hilfreich bei verschiedenen Histogrammen mit Fehlerbalken
 - `myH1->SetMarkerColor(3);`
 - `myH1->SetMarkerStyle(20);`
- Natürlich gehört zum jedem Histogramm auch eine sinnvolle Achsenbeschriftung:
 - `myH1->GetXaxis()->SetTitle("x-axis title");`
 - `myH1->GetYaxis()->SetTitle("y-axis title");`
- hierbei lassen sich an Latex angelehnte Kommandos verwenden
 - `myH1->GetXaxis()->SetTitle("p_{T} (GeV)");`
- der in Latex verwendete "`\`" wird in ROOT durch eine "`#`" ersetzt:

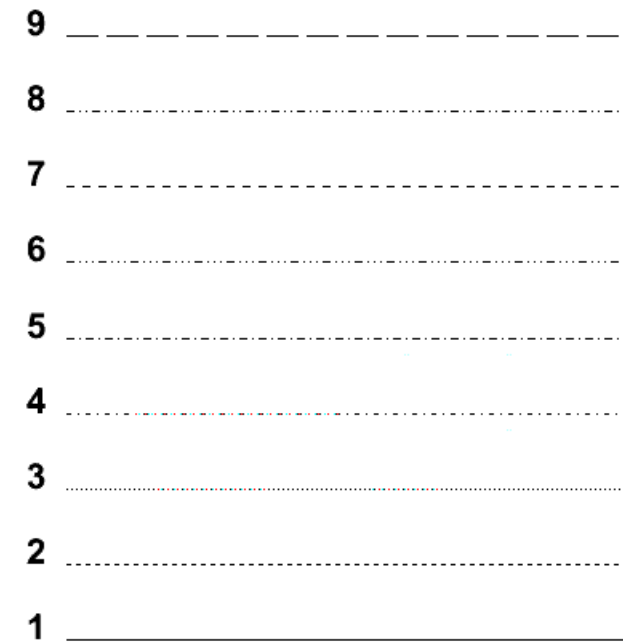
Bsp: `myH1->GetXaxis()->SetTitle("angle #phi (rad)");`

Optionen zum Zeichnen von Histogrammen

Farbpalette



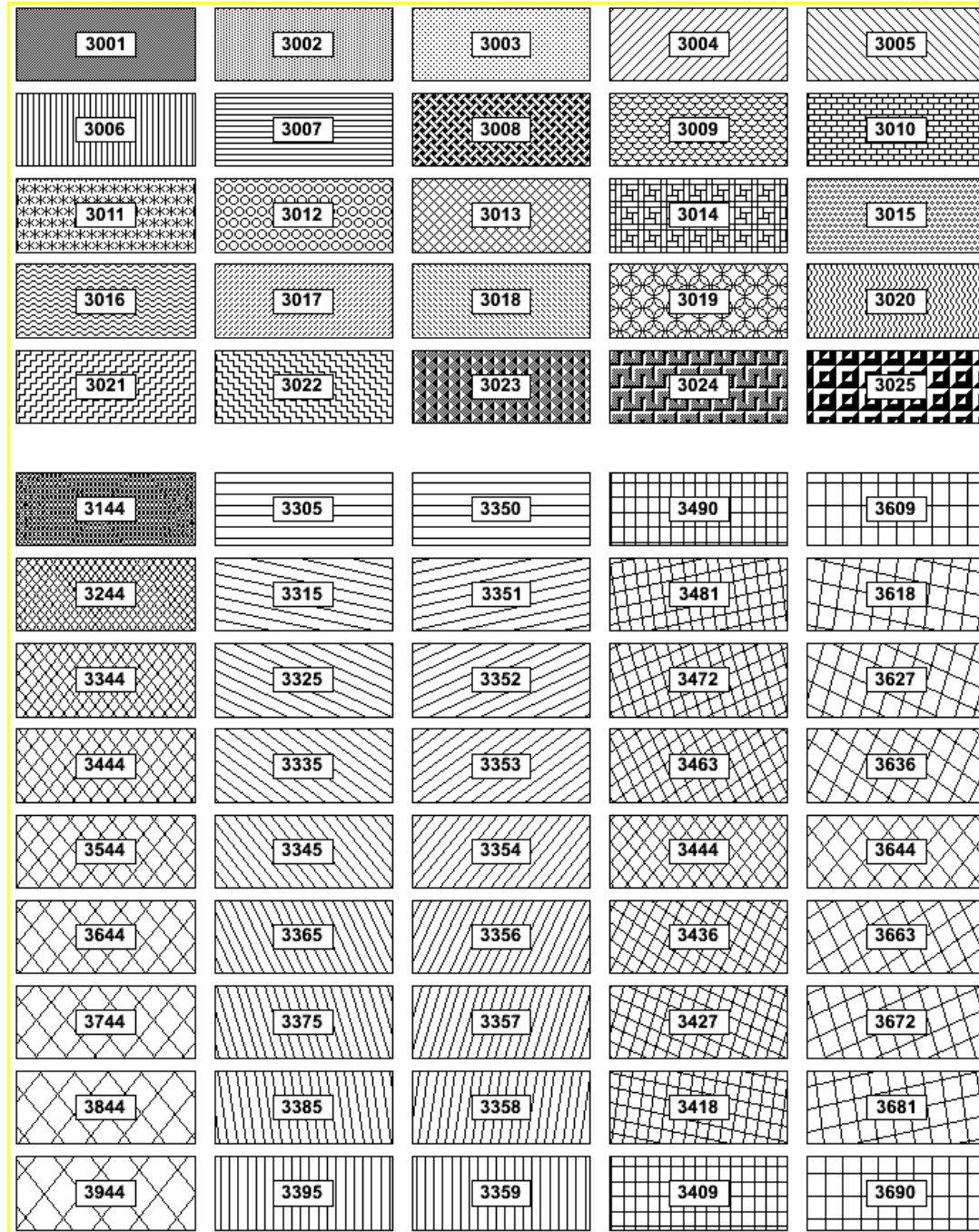
Line Styles



Marker Styles



Optionen zum Zeichnen von Histogrammen



Füllmuster

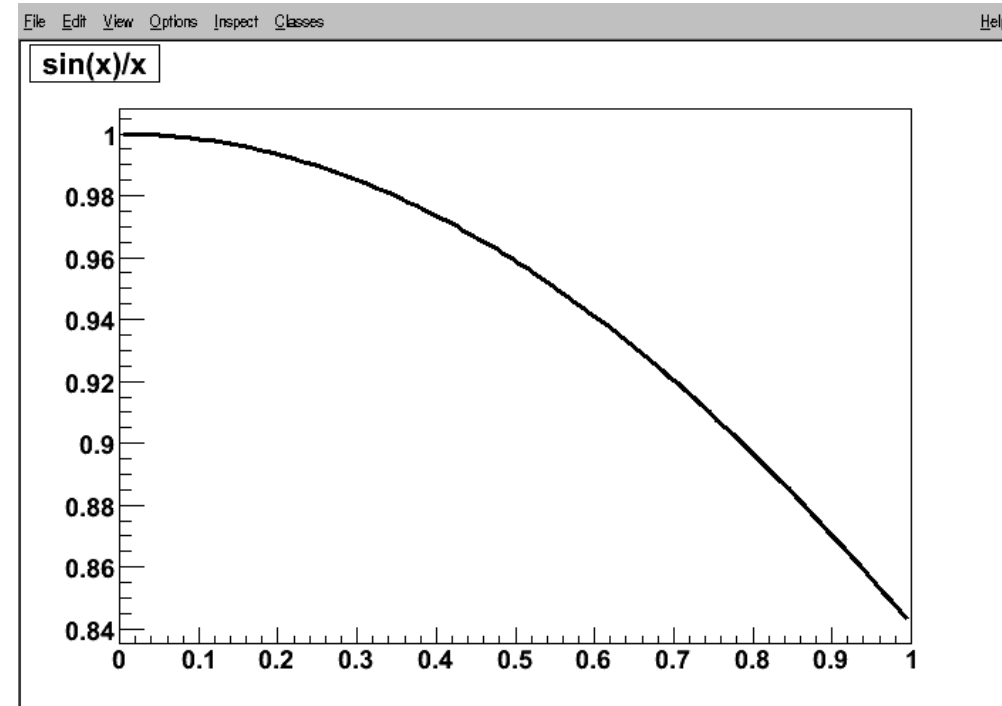
Funktionen (mat.)

- Ebenfalls wie Histogramme können auch Funktionen mit ROOT gezeichnet werden. Das folgende Macro zeichnet die Funktion $\sin(x)/x$ im Wertebereich von 0 bis 1.

```
//function.C
{
  gROOT->Reset();
  gROOT->SetStyle("Plain");
  TF1 *myFunc = new TF1("myFunction", "sin(x)/x", 0., 1.);
  myFunc->Draw();
}
```

Name Funktion Wertebereich

- Diese Funktionen werden uns später zum Fitten sehr hilfreich sein.



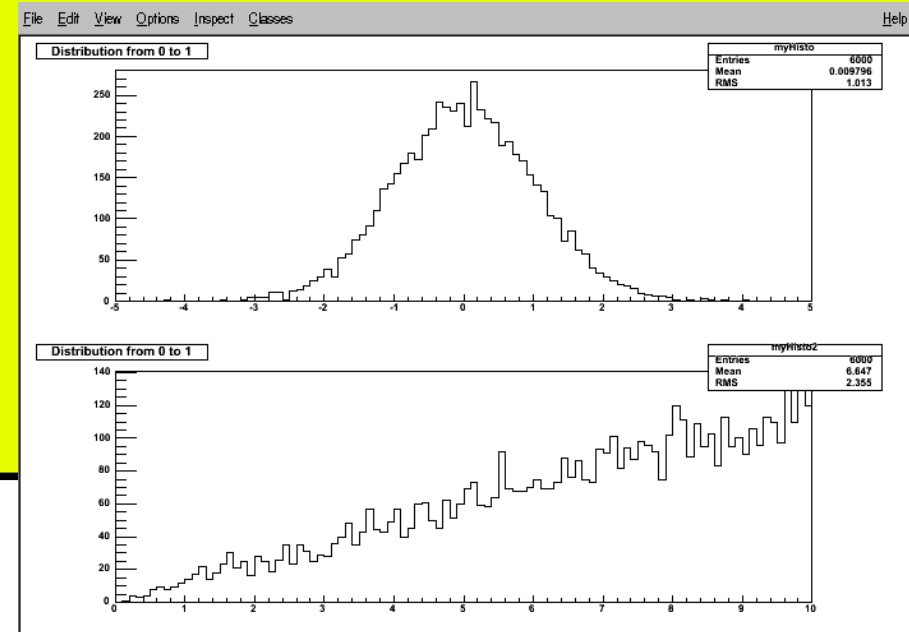
Funktionen (mat.)

- ROOT gibt bereits viele Funktionen vor die in `TMath` definiert sind. Diese sind gerade zum Fitten von Daten sehr nützlich. Eine kleine Auswahl:
 - `pol1, pol2, pol3....`
 - `gaus`
 - `Landau`
 - `BreitWigner`
 - `sin, cos, ...`
 - `sqrt`
 - `exp`
 - `log`
 - `...`
- Funktionen können auch kombiniert werden, dies ist gerade zum Fitten sehr nützlich. Hierbei lassen sich Anzahl der Parameter und Startwerte wählen. Weitere Infos:
 - <http://root.cern.ch/root/html402/TFormula.html>

Canvas

- Das Canvas (dt. Leinwand) stellt das Fenster dar, in welches Histogramme, und Graphen eingezeichnet werden. Bei Aufruf der `Draw()` Option wird ein Canvas automatisch erzeugt. Kreiert man ein Canvas allerdings bereits vorher von Hand, lassen sich gewisse Änderungen vornehmen. Möchte man z.B. zwei Histogramme nebeneinander zeichnen.

```
//histogram2.C
{
  gROOT->Reset();
  gROOT->SetStyle("Plain");
  TH1F* myH1 = new TH1F("myHisto","Distribution from -5 to 5",100,-5.,5.);
  myH1->FillRandom("gaus",6000);
  TH1F* myH2 = new TH1F("myHisto2","lin .Distribution",100,0.,10.);
  TF1* f1=new TF1("f1","2*x",0,10);
  myH2->FillRandom("f1",6000);
  TCanvas* c1=new TCanvas("myCanvas");
  c1->Divide(1,2);
  c1->cd(1);
  myH1->Draw();
  c1->cd(2);
  myH2->Draw();
}
```



Fitten von Daten

- Das nächste Beispiel illustriert das Fitten von Daten unter Verwendung der `Fit()` Funktion. Hierzu wird eine geeignete Funktionen mit sinnvollem Wertebereich gewählt und diese an die Daten (Histogramm) gefittet.

```
// fit.C
{
  gROOT->Reset();
  gROOT->SetStyle("Plain");
  TH1F* myH1 = new TH1F("myHisto","gaussian distribution",100,-5.,5.);
  TF1* myGaus = new TF1("myGaus","gaus",-5,5);
  myH1->FillRandom("gaus",6000);
  myH1->SetMarkerColor(2);
  myH1->SetMarkerStyle(20);
  myH1->Fit("myGaus");
  myH1->Draw("E");
  cout<<" -----" <<endl;
  cout<<" chi2/dof: " << myGaus->GetChisquare()/myGaus->GetNDF() <<endl;
  cout<<" mean: " << myGaus->GetParameter(1) <<" +/- " << myGaus->GetParError(1) <<endl;
  cout<<" width: " << myGaus->GetParameter(2) <<" +/- " << myGaus->GetParError(2) <<endl;
}
```

Fit-Bereich

Marker & Farbe ändern

Histogram nur als Datenpkt
mit Fehler zeichnen

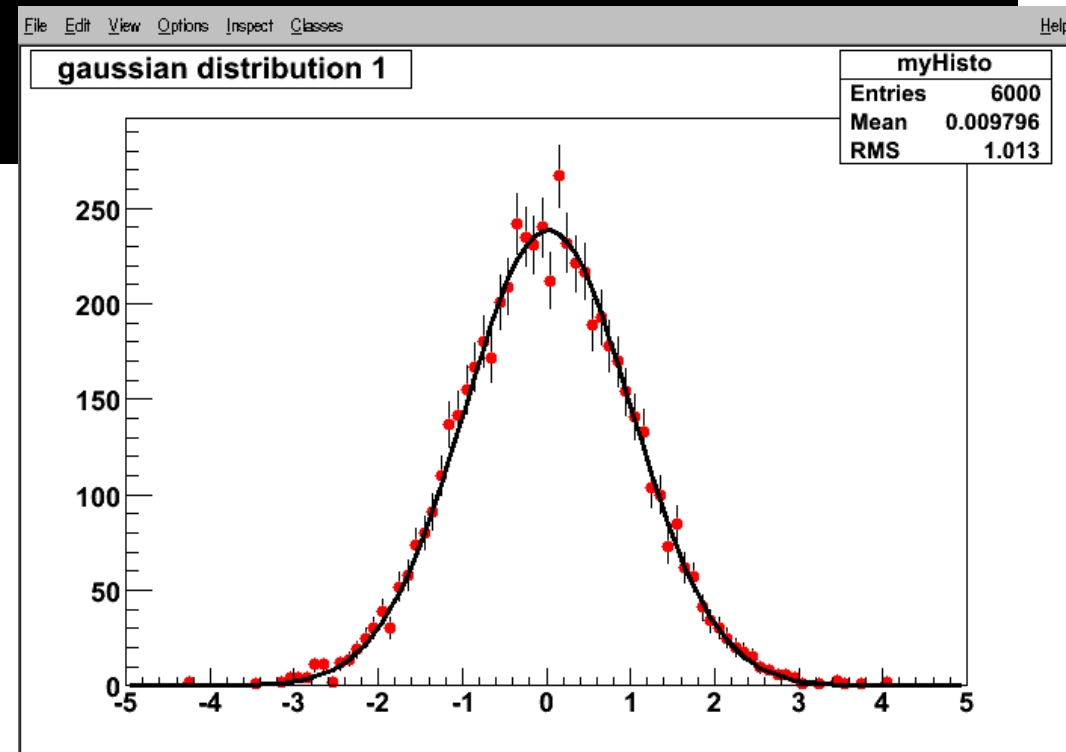
Ausgabe der Fit-Parameter und Güte
des Fits!

Fitten von Daten

- Ausgabe des Programms fit.C:

```
root [0] .x fit.C
FCN=65.2561 FROM MIGRAD      STATUS=CONVERGED      62 CALLS      63 TOTAL
                        EDM=1.03954e-09      STRATEGY= 1      ERROR MATRIX ACCURATE

EXT PARAMETER
NO.   NAME      VALUE      ERROR      STEP      FIRST
1   Constant    2.38701e+02    3.81138e+00    1.23151e-02    -6.02893e-06
2   Mean        2.07507e-02    1.29856e-02    5.15153e-05     2.51545e-03
3   Sigma       9.92900e-01    9.33484e-03    1.00065e-05     3.36906e-03
<TCanvas::MakeDefCanvas>: created default TCanvas with name c1
-----
chi2/dof: 0.973971
mean: 0.0207507+/-0.0129856
width: 0.9929+/-0.00933484
root [1]
```



Speichern von Histogrammen

- Eine weitere nützliche Funktion von ROOT ist das Speichern und Auslesen von Histogrammen in root-Files. Hierzu verwenden wir TFile Objekte:

- `TFile* _file=new TFile("file.root", "RECREATE")`

Dateiname

Option zum Öffnen der Datei

- ROOT schreibt das zu speichernde Histogramm mit in das letzte TFile-Objekt auf welches zugegriffen wurde:

- `histo->Write();`

- Vor Beenden des Macros muss die File von ROOT geschlossen werden:

- `file->Close();`

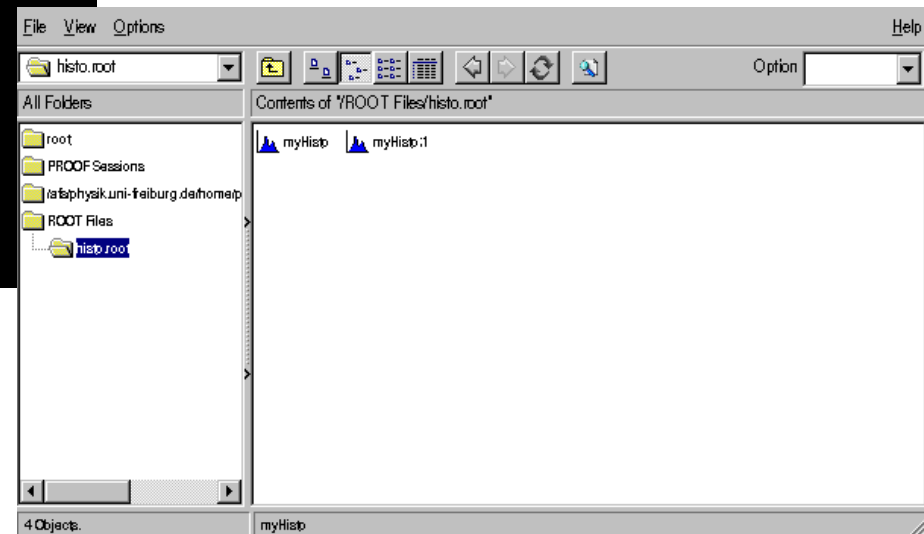
```
//saveHisto.C
{
  gROOT->Reset();
  gROOT->SetStyle("Plain");
  TH1F* myH1 = new TH1F("myHisto","gaussian distribution",100,-5.,5.);
  TFile* _file=new TFile("histo.root","RECREATE");
  myH1->FillRandom("gaus",6000);
  myH1->Draw();
  myH1->Write();
  _file->Close();
}
```

Lesen von Histogrammen

- In ROOT Files gespeicherte Objekte lassen sich entweder manuell mittels des `TBrowser`s betrachten oder in einem Macro auslesen und weiterverwenden:
- Zum manuellen Betrachten verwendet man den `TBrowser`, einem interaktiven Tool um u. A. den Inhalt von Root-Dateien zu browsen
- Um die Datei von Interesse direkt in ROOT zu laden einfach den Dateinamen als Argument an das shell-Kommando uebergeben:
 - `#>root filename.root`

```
penning@haco05:~fp/root/myprogs>root histo.root
root [0]
Attaching file histo.root as _file0...
root [1] new TBrowser
(class TBrowser*)0x8cd8950
```

starten des TBrowser



Lesen von Objekten

- Oft ist es praktischer ein gespeichertes Objekt wieder direkt in ein Macro zu laden und in diesem weiter zu verwenden.

```
//getHisto.C
{
  gROOT->Reset();
  gROOT->SetStyle("Plain");
  TFile* _file=new TFile("histo.root","OPEN");
  TH1F* _myH1 = (TH1F*)_file->Get("myHisto");
  _myH1->Draw();
}
```

- Dieses Prozedur aus

`TObject->Write()`

und

`TFile(TObject*)->Get("objName")`

lässt sich ebenfalls für andere ROOT-Objekte wie z.B. Funktionen TF1 verwenden

Einlesen von Daten aus ASCII-Dateien

- Um Daten aus dem FP die in ASCII-Files gespeichert sind einzulesen, können wir so vorgehen:

```
// readFile.C
{
  gROOT->Reset();
  gROOT->SetStyle("Plain");
  #include "Riostream.h"
  ifstream in;                      // Erzeugen des ifstream-Objektes
  in.open("peaks.dat");             // Oeffnen der File
  Float_t xi;
  Int_t nlines = 0;
  TFile* _file = new TFile("readData.root","RECREATE");
  TH1F* _histo = new TH1F("_histo","Peaks", 1250, 0., 1250 );

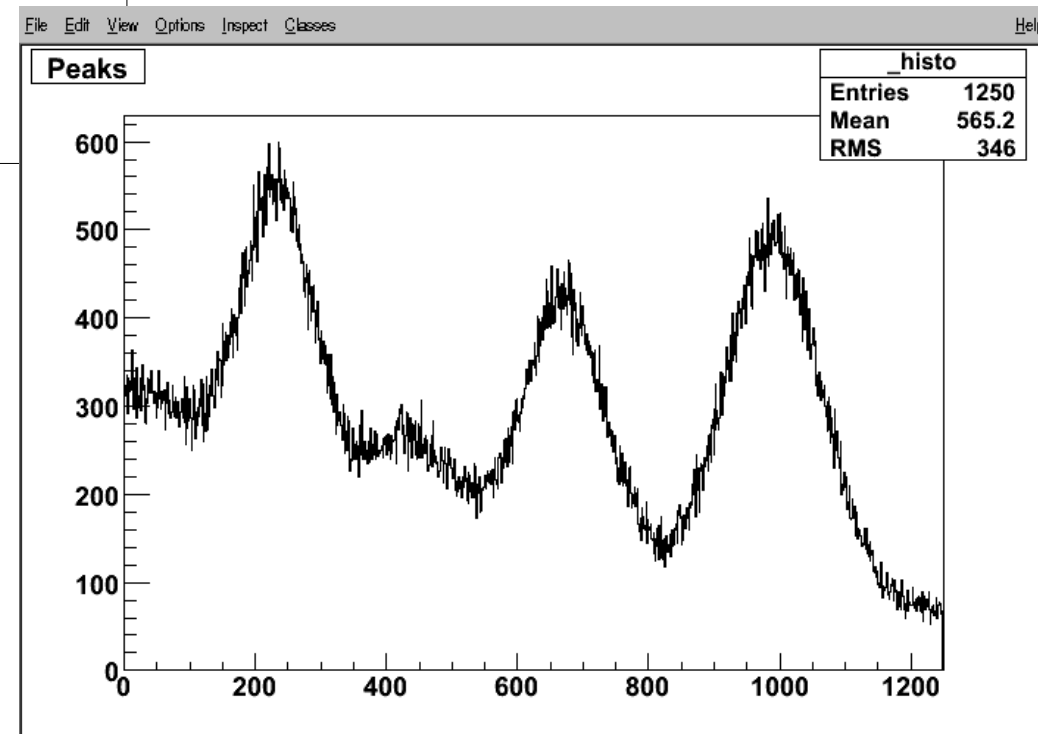
  while (1) {
    in >> xi;                      // fuellen des Wertes pro Zeile in Variable
    if (!in.good()) break;         //Abbrechen wenn Dateiende erreicht
    _histo->SetBinContent( nlines, xi ); //Fuellen des Histogramms, jede
    nlines++;                      // Zeile i entspricht Bin i
  }
  cout<<"found "<<nlines<<" data points"<<endl;
  in.close();
  _histo->Draw();
  _file->Write();
}
```

Einlesen von Daten aus ASCII-Dateien

- In diesem Fall ist die ASCII-File einspaltig:

```
322  
323  
322  
312  
314  
335  
291  
331  
329  
...
```

- Das Ergebnis des Programms:



Ein etwas komplizierteres Fit-Beispiel

- Im folgenden Beispiel fitten wir einen Gauß-Peak, der auf einem linear abfallenden Untergrund sitzt. Hierzu wird als Funktion die Summe aus einer Gaus-Funktion und einer linearen Funktion verwendet.

```
TF1* fitFunc = new TF1("fitFunc","pol1(0)+gaus(2)",0,300)
```

- Hierbei entspricht

$$\begin{aligned}\text{pol}(0)+\text{gaus}(2) &= [0]+[1]\cdot x+[2]\cdot \exp(-0.5*((x-[3])/[4])**2) \\ &= a+b\cdot x+c\cdot e^{-\frac{(x-d)^2}{2\cdot e}}\end{aligned}$$

- In dem Ausdruck "pol1(0)+gaus(2)" gibt (0) bzw. (2) den Beginn der Indizierung der Parameter für die entsprechende Formel an.
- Nützliche Funktionen:
 - func->SetParameter(Index,Wert);
 - func->SetParLimits(Index, Untergrenze, Obergrenze);
 - gStyle->SetOptFit();

Ein etwas komplizierteres Fit-Beispiel

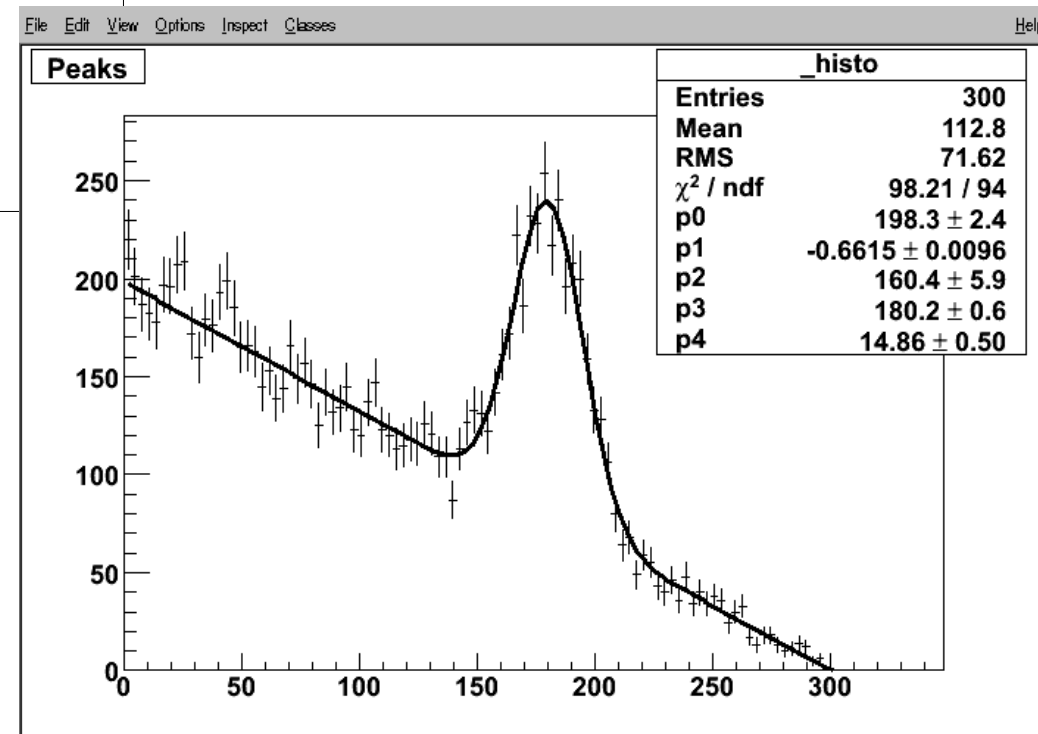
```
// fit2.C
{
  gROOT->Reset();
  gROOT->SetStyle("Plain"); // Ausgabe der Fit-Parameter in der Statistik-Box
  gStyle->SetOptFit();
  #include "Riostream.h"
  ifstream in;
  in.open("peak.dat");
  Int_t xi;
  Float_t yi;
  Int_t nlines = 0;
  TH1F* _histo = new TH1F("_histo","Peaks", 350, 0., 350 );
  while (1) {
    in >> yi >> xi; // Lese 2 Werte aus Datei ein
                     // setze den Wert von Bin  $x_i$  direkt
    if (!in.good()) break;
    _histo->SetBinContent( yi, xi );
    nlines++;
  }
  cout<<"found "<<nlines<<" data points"<<endl;
  TF1* fitFunc = new TF1("fitFunc","pol1(0)+gaus(2)",0,300);
  fitFunc->SetParameter(3,175); // Setzen sinnvoller Startwerte für
                                // Mittelwert & Breite des Gaus
  fitFunc->SetParameter(4,20);
  in.close(); // ändern des Binnings des Histograms
  _histo->Rebin(3);
  _histo->Fit("fitFunc");
  _histo->Draw("E");
}
```

Ein etwas kompliziertes Fit-Beispiel

- In diesem Fall ist die ASCII-File zweispaltig:

```
0      0
1     63
2     79
3     78
4     66
5     65
6     70
7     70
8     58
9     59
...
```

- und die Ausgabe



Kovarianz-Matrix

- Um die Kovarianz-Matrix eines Fits zu erhalten geht man folgendermassen vor:

```
TVirtualFitter *fitter = TVirtualFitter::GetFitter();  
TMatrixD *matrix = new TMatrixD(2,2,fitter->GetCovarianceMatrix());  
matrix->Print();
```

← Zugriffe auf das Fitter-Objekt
← Zugriffe auf die Kov.-Matrix
← Dim. der Matrix

- Bitte beachten, dass beim Initialisieren der Kovarianz-Matrix die korrekte Dimension anzugeben: $(n \times n)$, n = Anzahl der Fit-Parameter

```
{//linRegression.C  
  gROOT->Reset();  
  gROOT->SetStyle("Plain");  
  gStyle->SetOptFit();  
  TH1F* myH1 = new TH1F("myHisto","lin. regression",10,0.,10.);  
  TF1* myPol1 = new TF1("myPol1","2*x",0.,10.);  
  myH1->FillRandom("myPol1",1000);  
  myH1->Scale(0.1);  
  myH1->SetMarkerColor(2);  
  myH1->SetMarkerStyle(20);  
  myH1->Fit("pol1");  
  myH1->Draw("E");  
  TVirtualFitter *fitter = TVirtualFitter::GetFitter();  
  TMatrixD *matrix = new TMatrixD(2,2,fitter->GetCovarianceMatrix());  
  matrix->Print();  
}
```

← Fit mit Polynom 1. Grades, also lin. Regression

Kovarianz-Matrix

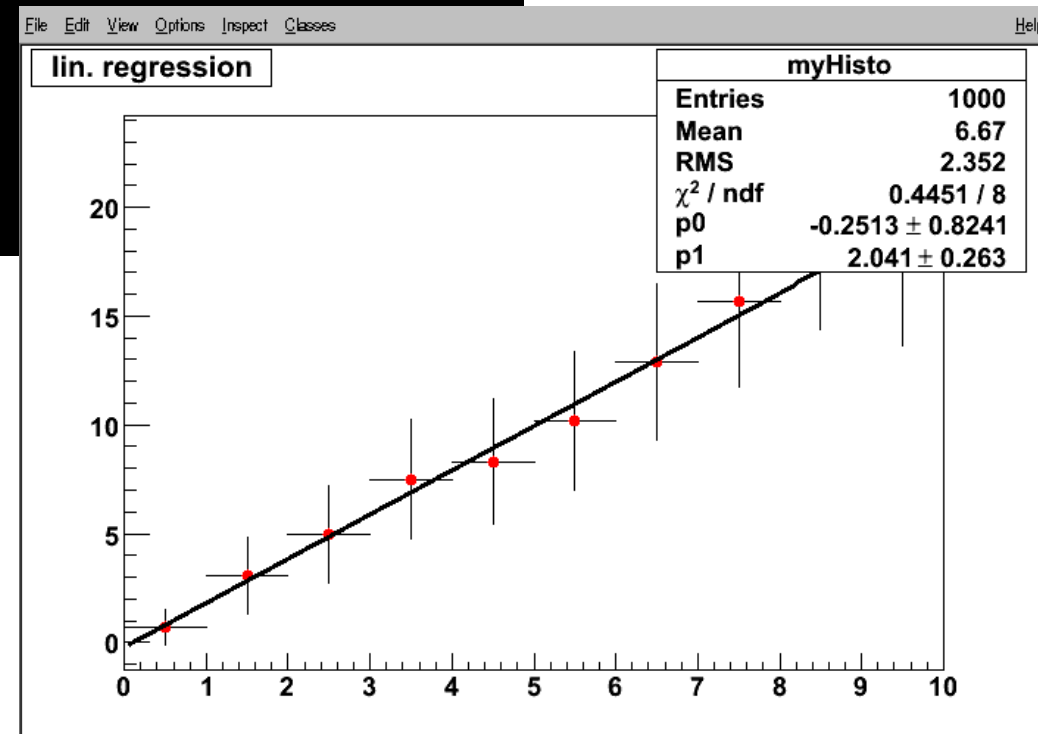
- Ausgabe des Programms:

```
root [0]
Processing linRegression.C...
Fitting results:
Parameters:
NO.          VALUE          ERROR
0          -2.513082e-01    8.241217e-01
1           2.041359e+00    2.626775e-01
<TCanvas::MakeDefCanvas>: created default TCanvas with name c1

2x2 matrix is as follows
```

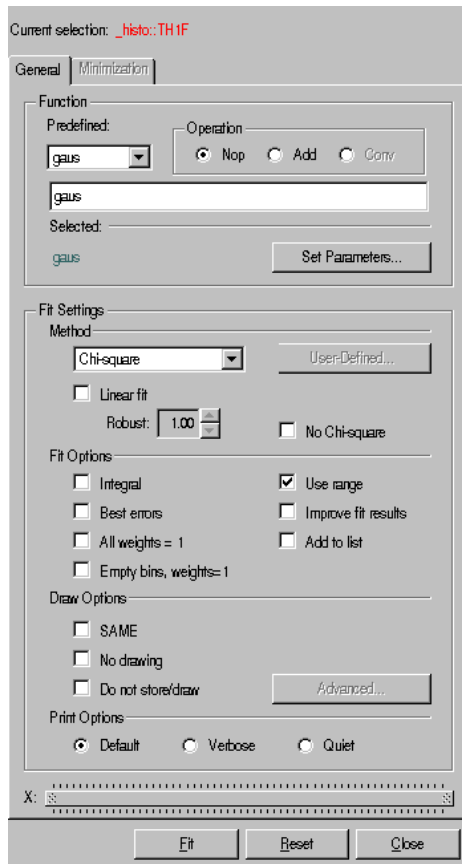
	0	1
0	0.6792	-0.1409
1	-0.1409	0.069

- Und der Fit

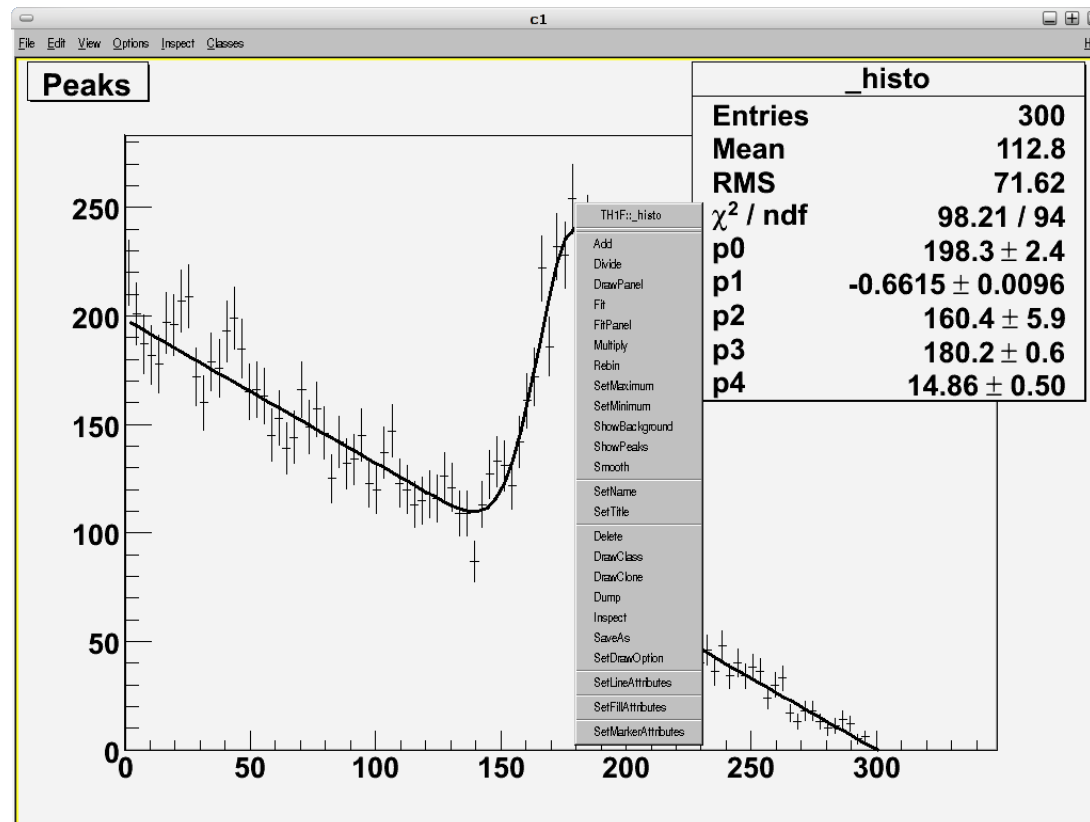


Interaktive Elemente

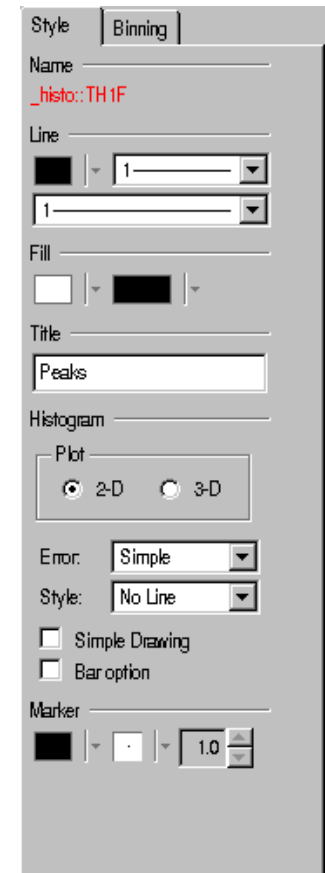
- ROOT erlaubt es auch, beinahe alle Operationen interaktiv über Menüs und Pull-Down Menüs auszuführen. Hierzu einfach mit der rechten Maustaste das gewünschte Objekt im Canvas anklicken und den Menüpunkt auswählen.



Fitpanel



Interaktive Elemente für Histogramme



Drawpanel

2 dimensionale Histogramme

- ROOT bietet ebenfalls die Möglichkeit, mehrdimensionale Histogramme oder Funktionen zu definieren. Ein zweidimensionales Histogramm bzw. Eine zweidimensionale Funktion werden wie folgt erzeugt.
 - `TH2F* _myH2 = new TH2F("my2Dhisto","2d histo",100,0.,1.,100,.0.,1.);`
 - `TF2* _myFunc2 = new TF2("my2Dfunc","2d func","x^2+y^2",-1.,1.,-1.,1.);`
- Die `Draw()` Methode fuer 2 dim. Histogramme und Funktionen bietet viele weitere zusätzliche Optionen. Optionen fuer 2 dim. Darstellen und solche fuer 3. dim. Darstellen. Beispielweise:
 - 2D: "BOX", "COL", "COLZ", "TEXT", "CONTO", "CONT1", "CONT2", "CONT3", "CONT4"
 - 3D: "LEGO", "LEGO1", "LEGO2", "SURF", "SURF1", "SURF2", "SURF3", "SURF4"
- Kleiner Tip, die Standard-Root-Farbpalette ist nicht sehr gelungen. Um das ansehnlicher zu gestalten sollte vor dem Zeichnen folgender Befehl ausgeführt werden:
 - `gStyle->SetPalette(1);`

2 dimensionale Histogramme

- Das folgende Beispiel zeigt kurz mehrdimensionale Funktionen und Histogramme und ebenfalls einige Darstellungsoptionen:

```
// 2d_histos.C
{
  gROOT->Reset();
  gStyle->SetPalette(1);
  TCanvas *c1 = new TCanvas("c1","Canvas fuer viele Histogramme",800,800);
  c1->Divide(2,2);
  TH2F *h2 = new TH2F("h2","Energie vs Impuls",40,-5.,5.,40,-5.,5.);
  h2->FillRandom("gaus",6000);
  h2->GetXaxis()->SetTitle("Energie E (GeV)");
  h2->GetYaxis()->SetTitle("Impuls p (GeV)");
  h2->GetZaxis()->SetTitle("Ereignisse");
  TF2* f2=new TF2("func2","sin(x)*sin(y)/(x*y)",-10.,10.,-10.,10.);
  c1->cd(1);
  h2->Draw("LEGO2");
  c1->cd(2);
  h2->Draw("COL");
  c1->cd(3);
  f2->Draw("SURF1");
  c1->cd(3);
  f2->Draw("SURF1");
  c1->cd(4);
  f2->Draw("COLZ");
}
```

2 dimensionale Histogramme

- Ausgabe:

