

Teilchenphysik Softwarekurs –Übungen–

Günter Duckeck
Johannes Elmsheuser
Alexander Mann
Sascha Mehlhase
Sommersemester 2015

Inhalt:

- Ein/Ausgabe von Daten
- Graphische Darstellung
- ATLAS ATLANTIS Event Display
- ROOT tuples und eine kleine Analyse
- Fitten von Daten
- HistPresent

Für die Übungen wird das `ROOT` Programmpaket verwendet, das eine umfangreiche Klassen- und Funktionenbibliothek in C++ bereitstellt.

Allgemeines

Kursziele:

- Einführung in und Arbeit mit ROOT
- Ereignisse des ATLAS Experiments visualisieren
- Einfaches Gerüst für eine Datenanalyse

Folien & Übungen (einseitig:ps,pdf, vierseitig:ps,pdf) im WWW

<http://www.etp.physik.uni-muenchen.de/kurs/comp14/>

1 Einführung in Linux und C/C++

1.1 Links zu C/C++ und Linux Tutorials

Eine kompakte Einführung in C/C++ gibt es unter:

- [C++ für Physiker](#)

Einige Tutorials zu Linux gibt es z.B. unter:

- [SelfLinux](#)
- [UNIX Tutorial for Beginners](#)

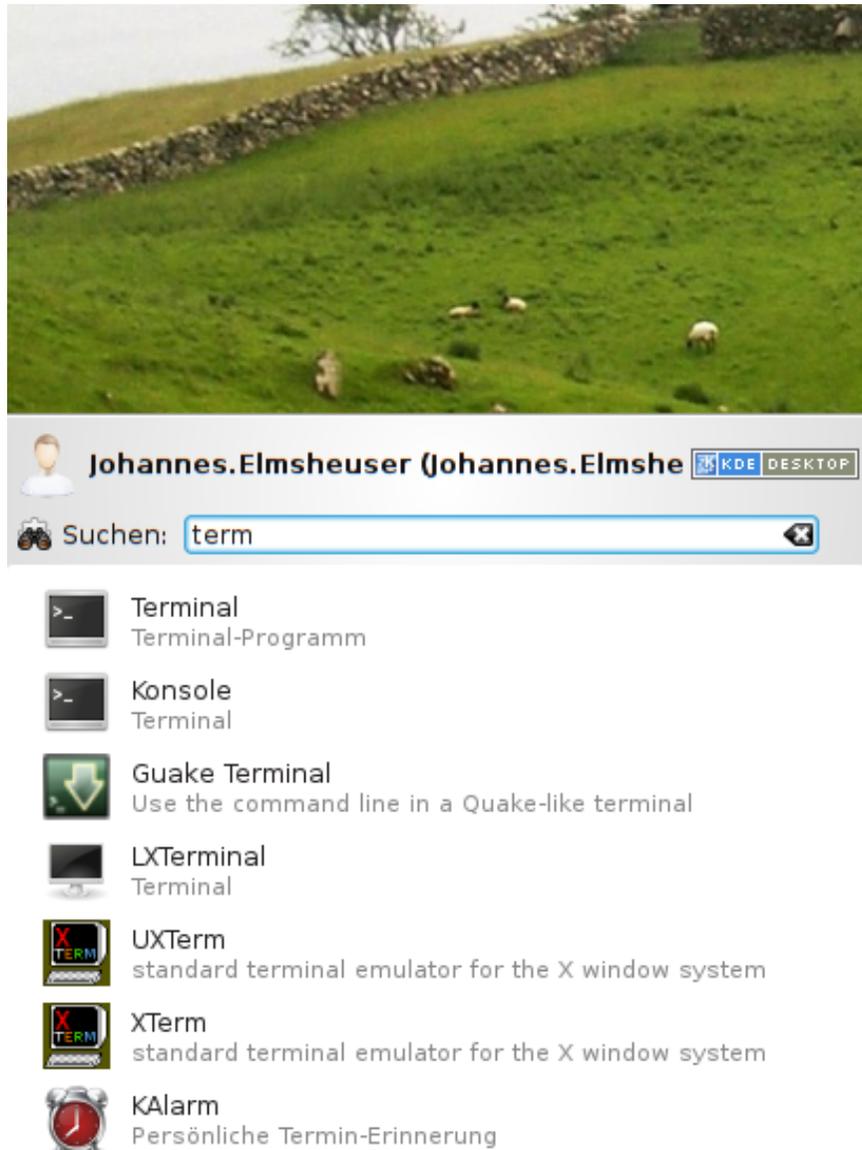
1.2 Die Linux X-Benutzeroberfläche

Die beliebtesten Benutzeroberflächen bzw. Fenstermanager auf Linuxsystemen sind: KDE bzw. GNOME. Diese können auf dem login-screen unter "Menü" -> "Session type" zwischen verschiedenen Fenstermanager aussuchen. Wählen Sie entweder "[KDE](#)" bzw. "[GNOME Classic](#)".

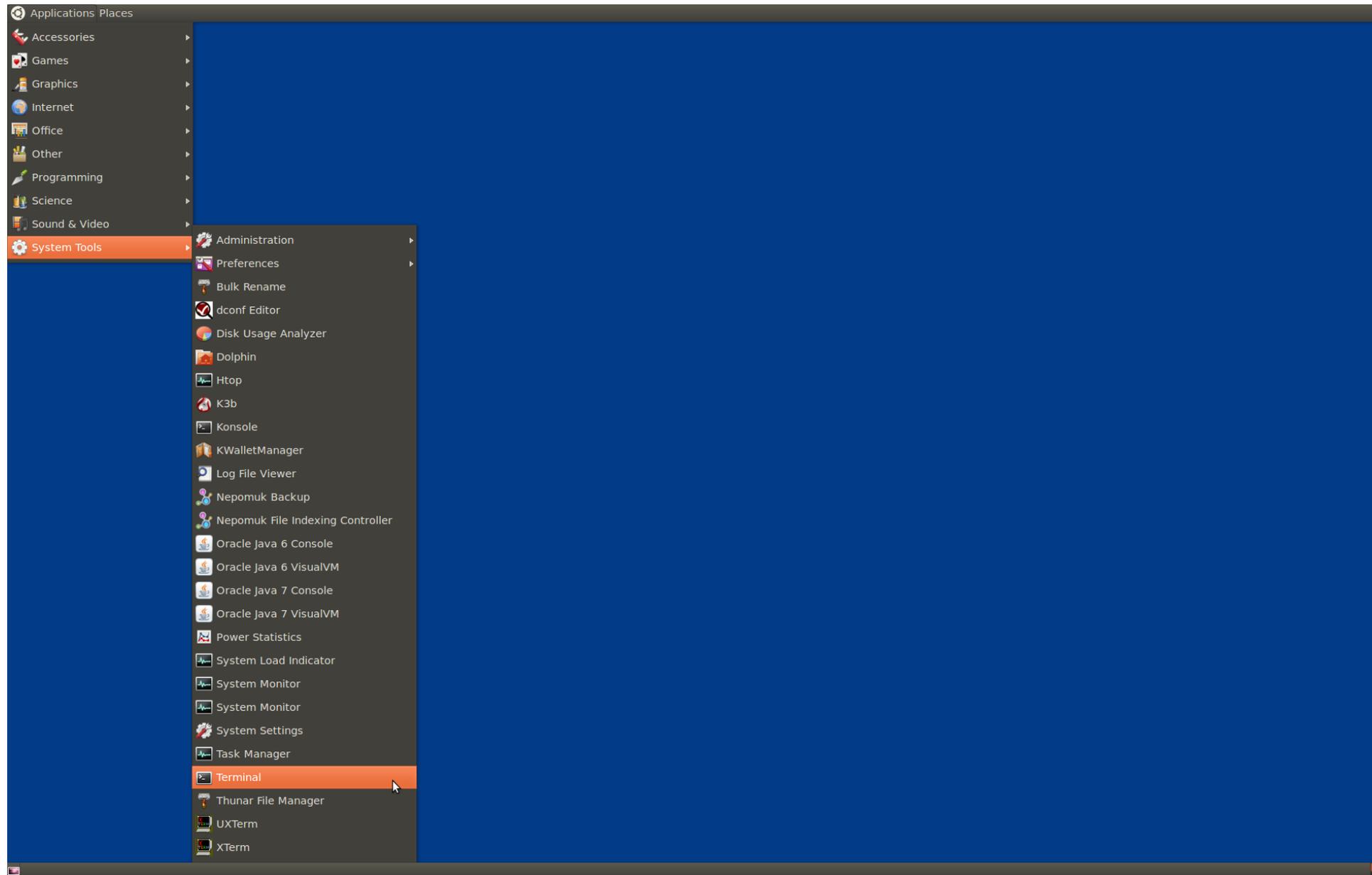
1.3 Terminalfenster starten

Nachdem Sie eingeloggt sind, starten Sie ein Terminalfenster, mit dem Sie verschiedene Befehle auf der Kommandozeile eingeben können:

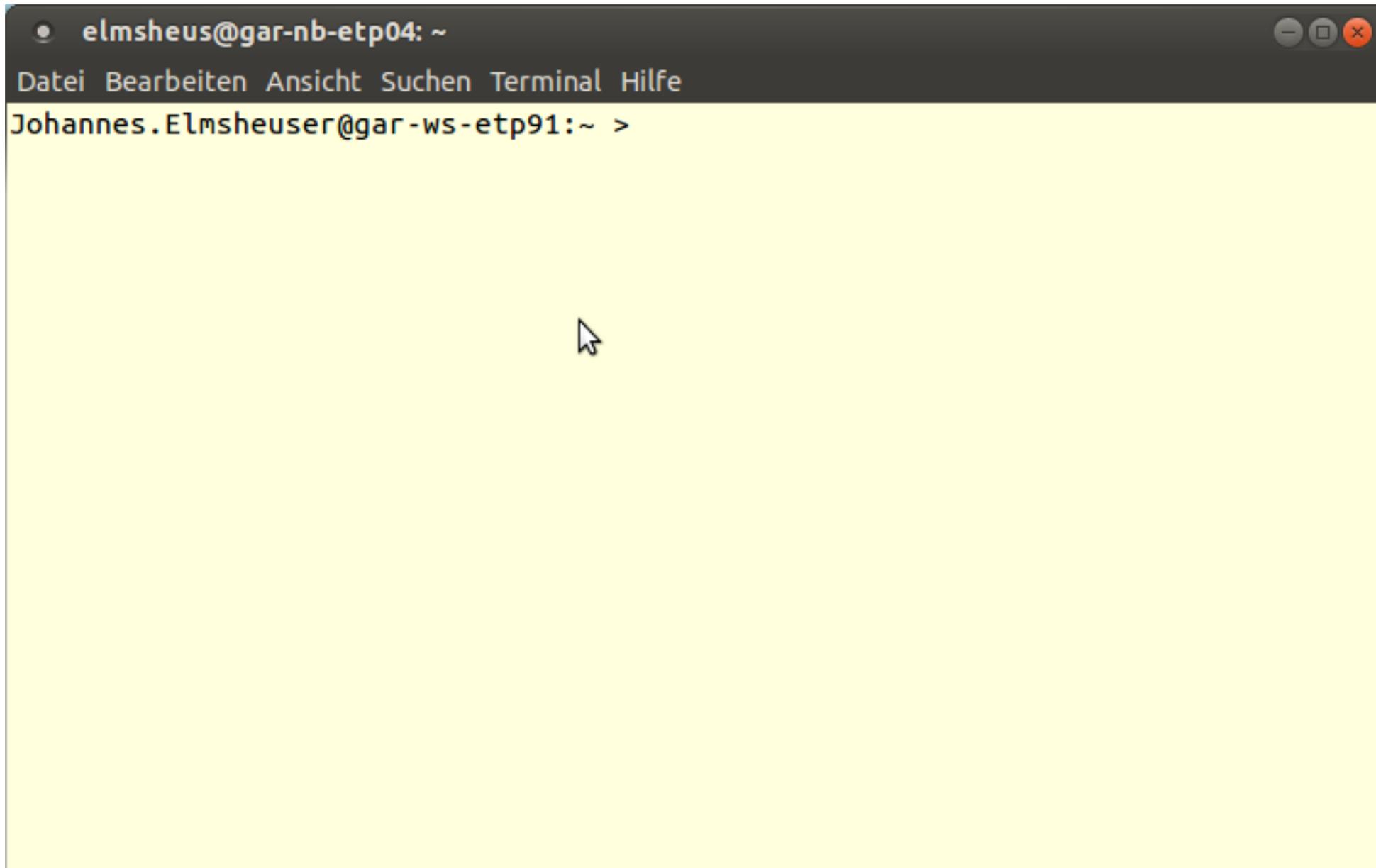
- Unter KDE klicken Sie in der rechten unteren Bildschirmecke auf den blauen "K"-Knopf und tippen in das danach erscheinende Suchfeld des Startmenüs: "term". Klicken Sie anschliessend auf den "Konsole" Menüeintrag und ein Terminalprogramm wird gestartet.



- Unter GNOME classic klicken Sie in der oberen rechten Bildschirmcke auf "Applications" und anschliessend im "System Tools"-Menü auf den Eintrag "Terminal":



- Es öffnet sich ein Terminalfenster, in dem Sie Befehle auf der sog. bash Kommandozeile eingeben und ausführen können:



The image shows a terminal window with a dark title bar and a light yellow background. The title bar contains the text "elmsheus@gar-nb-etp04: ~" and standard window control buttons (minimize, maximize, close). Below the title bar, a menu bar lists "Datei", "Bearbeiten", "Ansicht", "Suchen", "Terminal", and "Hilfe". The main area of the terminal displays the prompt "Johannes.Elmsheuser@gar-ws-etp91:~ >" and a mouse cursor is visible in the center.

```
elmsheus@gar-nb-etp04: ~
Datei Bearbeiten Ansicht Suchen Terminal Hilfe
Johannes.Elmsheuser@gar-ws-etp91:~ >
```

1.4 Befehle im Terminalfenster

Auf der bash Kommandozeile können Befehle eingegeben werden, um Programme zu starten oder z.B. mit dem Dateisystem zu interagieren.

Einfache Befehle:

- Das aktuelle Arbeitsverzeichnis anzeigen:

`pwd`

- Den Inhalt des aktuellen Verzeichnis anzeigen:

`ls`

- Den Inhalt des aktuellen Verzeichnis als Liste anzeigen:

`ls -l`

- Den Inhalt des aktuellen Verzeichnis als Liste mit versteckten Dateien anzeigen:

`ls -al`

- Den Inhalt des aktuellen Verzeichnis als Liste sortiert nach Änderungsdatum anzeigen:

`ls -rtl`

- In das Homeverzeichnis wechseln:

`cd`

- Ein neues Verzeichnis anlegen:
mkdir mycode
- In das neue Verzeichnis wechseln:
cd mycode

Weitere Befehle:

- Eine leere Datei anlegen:
touch test.txt
- Eine Datei löschen:
rm test.txt
- Eine Datei aus dem WWW herunterladen:
wget <http://www.etp.physik.uni-muenchen.de/kurs/Computing/ckurs/source/numbers.dat>
- Den Inhalt einer Datei vollständig anzeigen:
cat numbers.dat
- Den Inhalt einer Datei interaktiv anzeigen (Verlassen mit "q", Scrollen mit Pfeiltasten):
less numbers.dat
- Die Anzahl der Zeilen einer Datei anzeigen:
wc -l numbers.dat

- Ein leeres Verzeichnis löschen:
rmdir mytestdir
- Das aktuelle Verzeichnis kann mit "." angesprochen werden:
ls .
- In das übergeordnete Verzeichnis kann mit ".." angesprochen werden:
ls ..
- In das übergeordnete Verzeichnis wechseln:
cd ..
- Eine Datei von einem Verzeichnis in das aktuelle Verzeichnis kopieren:
cp /path/to/somefile .
- Eine Datei "somefile" vom Verzeichnis "/path/from" in das Verzeichnis "/path/to" kopieren:
cp /path/from/somefile /path/to/

Programme starten:

- Ein Programm starten Sie einfach durch Eingabe des Befehls auf der Kommandozeile. Dadurch wird die Kommandozeile für weitere Eingaben blockiert. Starten Sie deshalb sämtliche interaktiven Programme wie Editoren etc. immer mit einem zusätzlichem **&** am Ende der Befehlszeile, um die Kommandozeile wieder für neue Befehle freizugeben. Starten Sie den KDE Editor z.B. mit:
kate &

Befehlseingabe:

- Auf vorher eingegebene Befehle kann mit der Pfeil-nach-oben bzw. Pfeil-nach-unten Taste zugegriffen werden.
- Kommandozeilenvervollständigung: Lange Programmnamen können mit Hilfe der Tabulatortaste vervollständigt werden, d.h. Sie müssen nicht immer lange Programmnamen oder Dateinamen eintippen, sondern brauchen nur die Anfangsbuchstaben eintippen und nach Drücken der Tabulatortasten kann die Befehlszeile vervollständigt werden.

Eingabe-/Ausgabeumleitung:

- Die Ausgabe eines Programms oder eines beliebigen Befehls kann vom Bildschirm des Terminalfensters in eine Datei mit ";>" umgeleitet werden:

ls -rtl > out.txt

- Die Eingabe in ein Programm kann anstatt von der Tastatur von einer Datei mit "<|" umgeleitet werden:

cat <| numbers.dat

Editoren:

- KDE Editor:

kate

- GNOME Editor:

gedit

- Fortgeschrittene Editoren:

emacs oder **vi**

Entwicklungsumgebungen und Debugger:

- Java, C++, Python Entwicklungsumgebung:

eclipse

- C++ Entwicklungsumgebung:

kdevelop

- Qt und C++ Entwicklungsumgebung:

qtcreator

- Graphischer Debugger:

ddd

GNU C++ Compiler:

- Ein C++ Programm kompilieren und linken in einem Schritt:

g++ -o mytest mytest.cpp

- Ein C++ Programm kompilieren:

g++ -c mytest.cpp

- Ein zusammengesetztes C++ Programm kompilieren und linken:

g++ -o TLVector MyLVector.cpp My3Vector.cpp**Verzeichnisse archivieren:**

- Das aktuelle Verzeichnis in eine Datei archivieren und packen:

tar cvzf myfile.tar.gz .

- Ein Archivdatei in aktuelle Verzeichnis entpacken:

tar xvzf myfile.tar.gz

2 Einführung in ROOT

2.1 Was ist ROOT ?

Programmpaket zur Datennahme, Simulation und Datenanalyse, wird am CERN (European Center for Particle Physics, Genf) entwickelt.

- basierend auf C++
- objektorientiert

ROOT stellt eine Vielzahl von Objekten/Funktionen zur Verfügung

- Daten I/O
- und Verwaltung
- Simulation
- Analysieren
- graphische Darstellung

Wir beschränken uns auf einen kleinen Teilbereich:

- interaktives Arbeiten mittels C++ interpreter `CINT`
- C++ Makros
- Funktionen plotten
- Histogramme und Graphen
- Random numbers
- Fits

2.2 ROOT links:

- **Main ROOT page am CERN**
- **ROOT Tutorials**
- **ROOT Klassenindex**
- **ROOT User Guide**
- **Ausführliches Tutorial zur Benutzung von ROOT (PDF) bzw. mit Macros (ZIP)**

C++ :

- **C++ für Physiker (LMU)**
- **C++ basics for ROOT users**
- Nützlich auch die **Numerical Recipes in C**. (Nur alte Version frei verfügbar...)

2.3 Arbeiten mit ROOT

2.3.1 ROOT initialisieren und starten

Vor dem ersten Mal:

- Verzeichnis anlegen:

```
mkdir root
```

- Initialisierungsfiles kopieren:

```
cp /project/etp/rootcourse/macros/rootlogon.C .
```

-

ROOT starten:

```
// Pfade setzen (evt. in .bashrc kopieren)
module load marabou (oder: module load root)
cd root
root
```

ROOT beenden:

```
!q
```

2.3.2 C++ Operationen – ROOT als Taschenrechner

- alle C++ Operationen
 - arithmetisch: `+`, `--`, `*`, `/`
 - bitwise: `&`, `|`, `<<`, `>>`, `~`
 - logical: `&&`, `||`, `!`
- CINT zusätzlich Exponent (`**`)
- C++ standard math. Funktionen:
sin, cos, tan, atan, log, exp ...
- weitere Funktionen in **TMath** Klassenbibliothek

2.3.3 ROOT Datentypen

ROOT definiert (`typedef`) eigene Datentypen um einheitliche Byte-Länge auf unterschiedlichen Systemen zu gewährleisten.

C++	FORTRAN	Bytes	ROOT	Bytes
char	CHARACTER*1	1	Char_t	1
int	INTEGER	2/4	Int_t	4
long		4/8	Long_t	8
float	REAL*4	4	Float_t	4
double	REAL*8	8	Double_t	8

⇒ Wichtig für Datenaustausch zwischen unterschiedlichen Systemen.

```

root [0] 5**2                                     1
(int)25                                           2
root [1] 25.1*3.5                                 3
(double)8.785000000000000085e+01                 4
root [2] 25.1**3.5                                5
(double)7.92242296929665754e+04                  6
root [3] sin(2)                                   7
(double)9.09297426825681709e-01                  8
root [4] tan(1)                                   9
(double)1.55740772465490229e+00                  10

```

root [5] atan(1)	11
(double)7.85398163397448279e-01	12
root [6] atan(1)*4	13
(double)3.14159265358979312e+00	14
root [7] log(2)	15
(double)6.93147180559945286e-01	16
root [8] exp(10)	17
(double)2.20264657948067179e+04	18
root [9] exp(0)	19
(double)1.0000000000000000e+00	20
root [10] exp(1)	21
(double)2.71828182845904509e+00	22
root [14] 2<<10	23
(int)2048	24
root [15] 3<<10	25
(int)3072	26
root [16] sqrt(2)	27
(double)1.41421356237309515e+00	28
root [17] atan2(1,1)	29
(double)7.85398163397448279e-01	30

```
root [18] atan2(1,0) 31  
(double)1.57079632679489656e+00 32
```

2.3.4 Schleifen

```
root [40] Double_t s = 1; 1  
root [41] for ( Int_t i=1; i<70; i++ ) s *= i // Fakultaet 2  
root [42] s 3  
(Double_t)1.71122452428141297e+98 4  
root [57] TMath::Gamma(70) // Dasselbe mit Gamma Fkt 5  
(Double_t)1.71122452441969184e+98 6
```

2.3.5 Macros

```
// file fak.C 1
Double_t fak( Int_t n = 1 ) 2
{ 3
  Double_t t = 1; 4
  for ( Int_t i = 1; i <= n; i++ ) { 5
    t *= i; 6
  } 7
  return(t); 8
} 9
root [63] .x fak.C(99) // direkt ausfuehren 10
9.33262e+155 11
root [64] .L fak.C // oder erst Laden 12
root [65] fak.C(99) // dann aufrufen 13
9.33262e+155 14
```

2.3.6 Macro Variationen

In Root gibt es verschiedene Möglichkeiten Macros zu definieren und auszuführen

- Direktes Ausführen:

```
.x myMacro.C
```

Zwei Arten von Macros:

- **Named Macro:** .C-Datei enthält Funktion mit gleichem Namen. Bei Aufruf des Macros wird diese Funktion ausgeführt.

Z.B. `.x fillH1.C(10000)`

```
int fillH1( int n=100000 )           1
{                                     2
  TH1F * h2 = new TH1F("h2","mytitle",100,0,1); 3
  for ( Int_t i = 0; i<n; i++ ) {       4
    h2->Fill(gRandom->Rndm());          5
  }                                     6
  h2->Draw();                           7
}
```

8

Optional können Argumente übergeben werden.

Alle Variablen, die in der Funktion angelegt werden, sind nach Ausführung wieder verschwunden (*local scope*).

- **Un-Named Macro:** Keine Funktionsdeklaration, Root/C++ Anweisungen stehen direkt in Block von {}:

```
{
int n=100000;
TH1F * h2 = new TH1F("h2","mytitle",100,0,1);
for ( Int_t i = 0; i<n; i++ ) {
    h2->Fill(gRandom->Rndm());
}
h2->Draw();
}
```

1
2
3
4
5
6
7

8

Derselbe Effekt wie wenn diese Anweisungen direkt auf Root Kommandozeile eingegeben werden, insbesondere bleiben die Variablen erhalten (*global scope*)!

- Laden der Macro Datei:

- Laden mit Cint (C++ Interpreter)

```
.L myMacro.C
```

Datei wird von CINT interpretiert, es können mehrere Funktionen deklariert werden, Funktionsnamen beliebig, unabhängig von Dateiname.

- * Vorsicht mit CINT-spezifischen Eigenschaften, nicht 100% C++ kompatibel
- * Standard Root Klassen (TH1F) bekannt, kein `#include <TH1F.h>` nötig.
- * Variablen sind im (*local scope*).

Manchmal obscure CINT Fehlermeldungen und Abbrüche, praktisch für einfache, kurze Funktionen.

- Laden und Kompilieren mit C++ Compiler

```
.L myMacro.C+ bzw. .L myMacro.C++
```

Datei wird kompiliert und *shared-object library* erzeugt und dynamisch geladen (`myMacro_d.so`).

- * Im 1. Fall (.C+) wird Datei nur kompiliert wenn geändert seit letzter Erstellung der .so lib
- * Im 2. Fall (.C++) wird immer kompiliert.

```
#include <TH1F.h> 1
#include <TRandom.h> 2
void fillH1( int n=100000 ) 3
{ 4
```

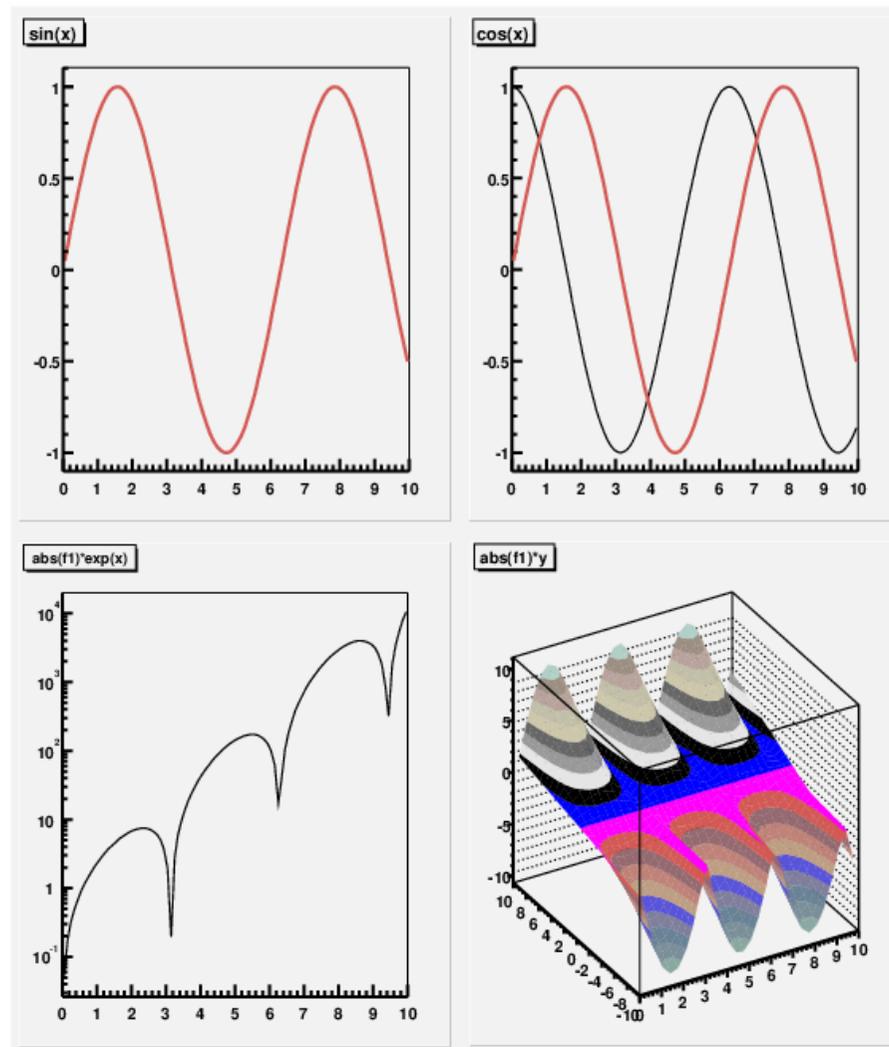
```
TH1F * h2 = new TH1F("h2","mytitle",100,0,1);           5
for ( Int_t i = 0; i<n; i++ ) {                         6
    h2->Fill(gRandom->Rndm());                           7
}                                                         8
h2->Draw();                                              9
}                                                         10
```

Alle verwendeten Klassen/Funktionen müssen über die entsprechenden Header Dateien deklariert werden!

Üblicher C++ Standard, sehr gute Performance.

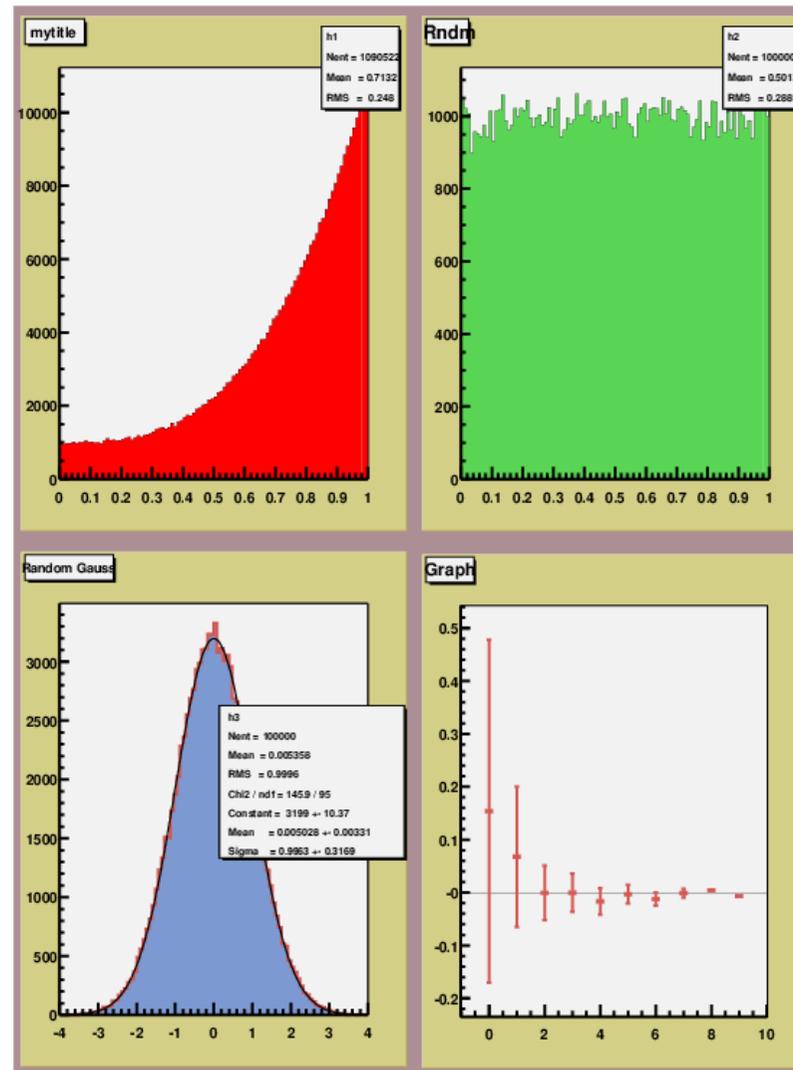
2.3.7 Funktionen

```
root [60] TF1 *f1 = new TF1("f1","sin(x)", 0, 10 );           1
root [61] f1->Draw();                                     2
root [68] TF1 *f2 = new TF1("f2","cos(x)", 0, 10 );       3
root [69] f2->Draw();                                     4
root [70] f1->Draw("same");                               5
root [76] TF1 *f4 = new TF1("f4","abs(f1)*exp(x)", 0, 10 ); 6
root [77] f4->Draw();                                     7
root [78] TF2 *ff1 = new TF2("f5","abs(f1)*y", 0, 10, -10, 10 ); 8
root [79] ff1->Draw()                                     9
```



2.3.8 Histogramme und Zufallszahlen

```
root [83] TH1F * h1 = new TH1F("h1","mytitle",100,0,1);           1
root [84] for ( Float_t x = 0; x<1; x += 1e-6 ) h1->Fill(x,x**3); 2
root [85] h1->Draw();                                           3
root [80] TH1F * h2 = new TH1F("h2","mytitle",100,0,1);       4
root [81] for ( Int_t i = 0; i<100000; i++ ) h2->Fill(gRandom->Rndm()); 5
root [82] h2->Draw();                                           6
root [87] TH1F * h3 = new TH1F("h3","Random Gauss",100,-4,4); 7
root [88] for ( Int_t i = 0; i<100000; i++ ) h3->Fill(gRandom->Gaus()); 8
root [89] h3->Draw();                                           9
```



2.3.9 Fitten

```
// file gauf.C 1
{ 2
  Int_t nit = 10, nrnd, i, j; 3
  Float_t mean[10], emean[10], xv[10], ex[10]; 4
  TH1F * h3 = new TH1F("h3","Random Gauss",100,-4,4); 5
  j =0; 6
  for ( i =0; i<nit; i++ ) { 7
    nrnd = 100*pow(2,i); 8
    xv[i] = i; 9
    ex[i] = 0.1; 10
    for ( ; j<nrnd; j++ ) { 11
      h3->Fill(gRandom->Gaus()); 12
    } 13
    h3->Fit("gaus","eq"); 14
    h3->Draw(); 15
    TF1 *fit = h3->GetFunction("gaus"); 16
    mean[i] = fit->GetParameter(1); 17
    emean[i] = fit->GetParError(1); 18
```

```
    cout << i << " Mean = " << mean[i] << " +- " << emean[i] << endl;    19
}                                                                              20
TCanvas *ce = new TCanvas("ce", "ce");                                       21
TGraphErrors *tg = new TGraphErrors( nit, xv, mean, ex, emean );            22
tg->Draw("AP");                                                                23
}                                                                              24
root [1] .x gausf.C                                                            25
```

2.3.10 NTuple und Tree Ausgabe

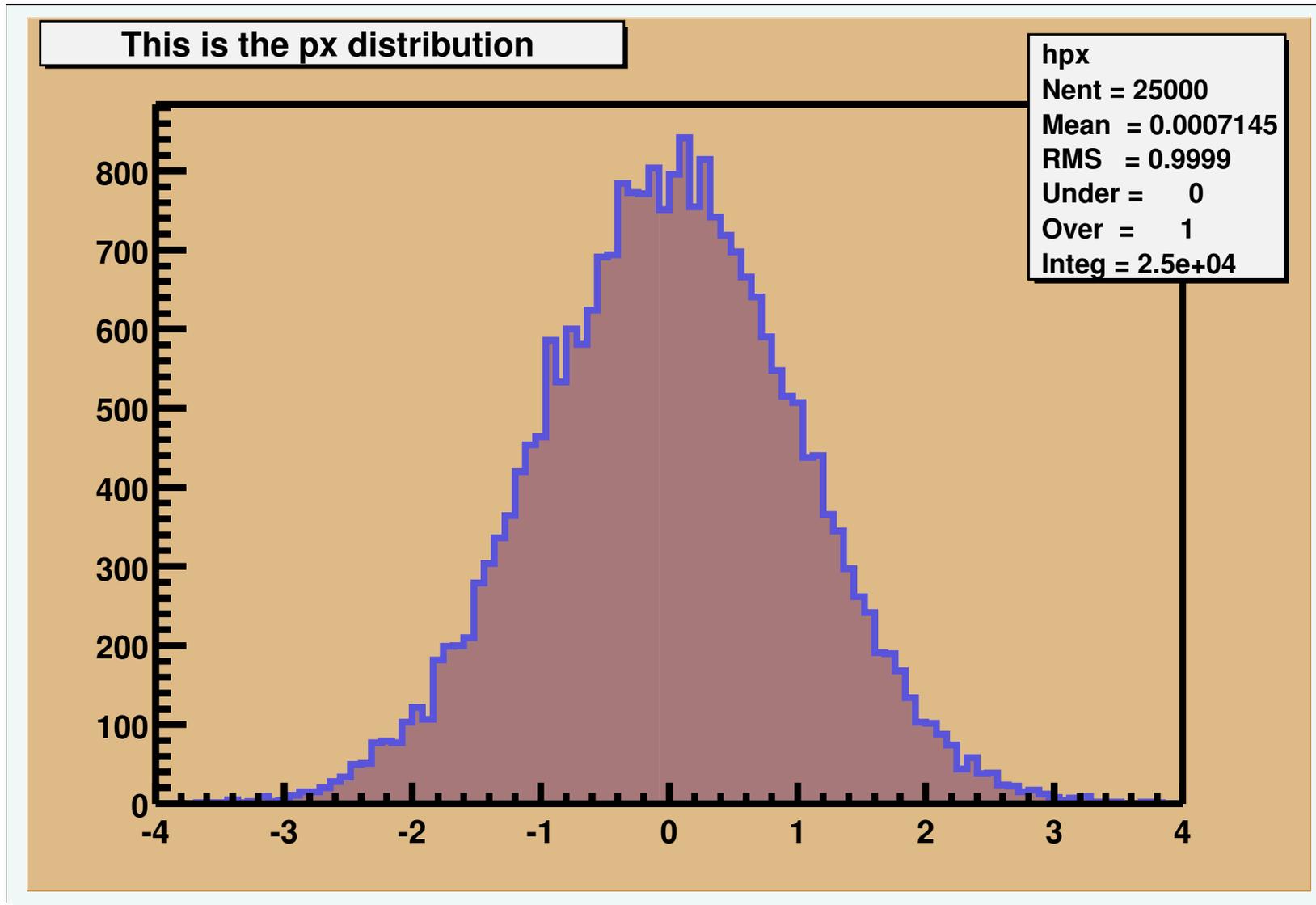
- `basic.C`
- `basic2.C`
- `basic.dat`

2.4 Datenvisualisierung

In Naturwissenschaften i.a. zwei Arten von Daten:

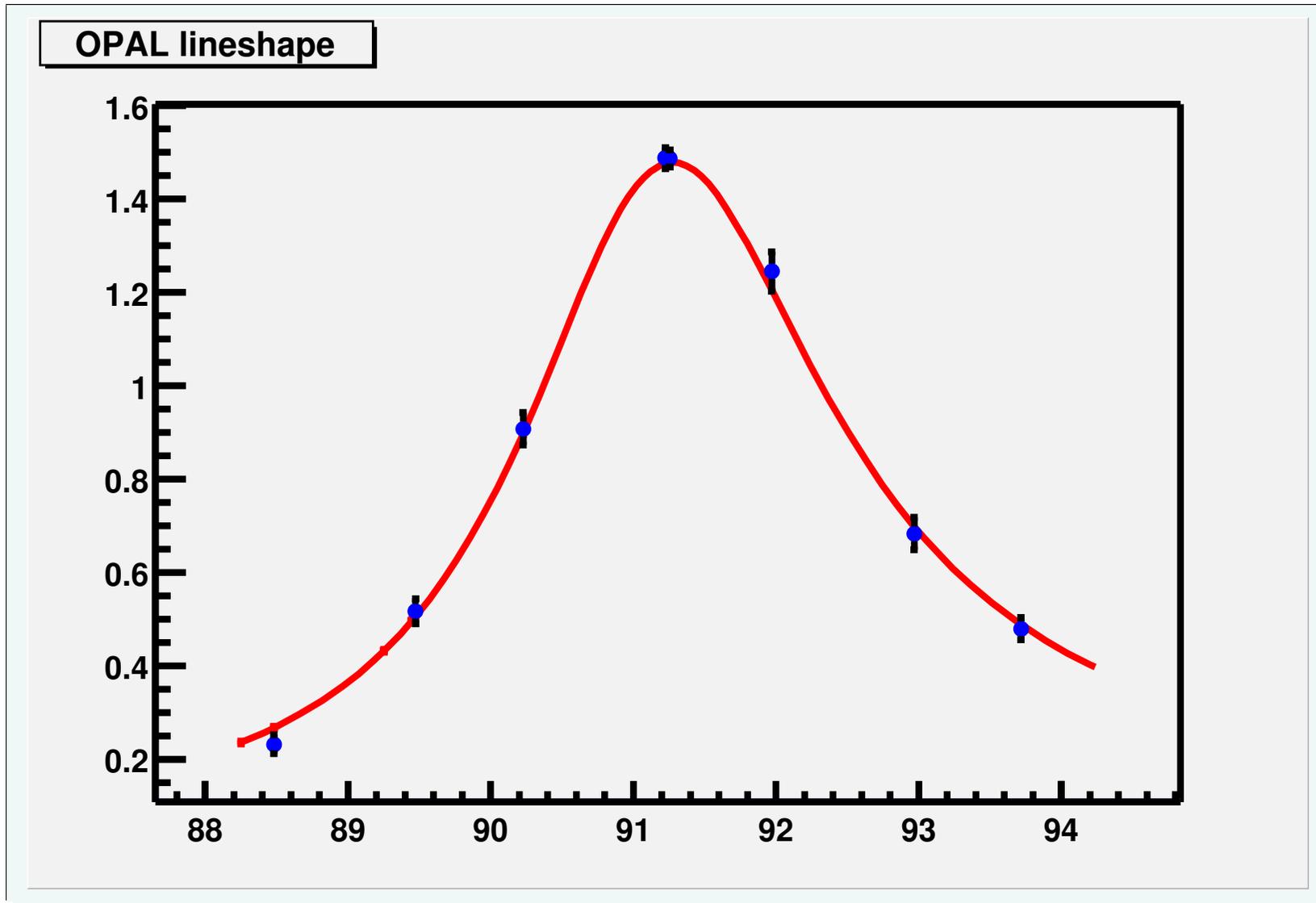
Häufigkeitsverteilung: Eine oder mehrere Größen werden wiederholt gemessen. Darstellung in ein- oder mehr-dimensionalen 'Histogrammen':

- Bereich `xlow` bis `xhigh` unterteilt in `nchannel` Intervalle.
- `TH1F * h1 = new TH1F("h1","mytitle",nchannel, xlow, xhigh)`
- Jede Messung `x` wird in das Histogramm gefüllt:
`h1->Fill(x)`
- Darstellung \Rightarrow Einträge pro Intervall.



(x,y) Wertepaare mit Fehlern: Messungen einer Grösse y in Abhängigkeit von x mit bekannten Fehlern Δy (und evt. auch Δx) werden in einen 'x,y-Graph' eingetragen.

- np Wertepaare x, y mit Fehlern ex, ey werden in ein Diagramm eingetragen
- `TGraphErrors *tg = new TGraphErrors(np, x, y, ex, ey)`
- `x, y, ex, ey` sind jeweils arrays vom Typ `double a[np]`



3 Datenanalyse – einfach

3.1 Einführung

ROOT ermöglicht die effiziente und schnelle Analyse von sehr großen Datenmengen. Daten können entweder in ASCII-Format gespeichert und mit normalen C/C++ Befehlen eingelesen und weiterverarbeitet werden. Sehr viel effizienter ist die Speicherung im ROOT-tuple- bzw. n-tuple-Format. In diesem Format werden Daten und ihre Eigenschaften bzw. Variablen in sog. Baum-Format (Trees) strukturiert. Root-Trees sind optimiert zum Speichern und effizienten Prozessieren von *Event-Daten* der Teilchenphysik.

Typischerweise werden *Events*, die im Detektor aufgezeichnet werden, in sehr unterschiedlichen Varianten abgespeichert und prozessiert.

- **Rohdaten** enthalten alle Detailinformationen der Sub-Detektoren, z.B. die Driftzeiten jedes Rohrs im Myinspektrometer, das ein Signal registriert hat. Das erfordert eine komplexe, tiefe Tree-Struktur:
ATLAS \Rightarrow Muonspektrometer \Rightarrow Kammer-ID \Rightarrow Rohr-ID \Rightarrow Driftzeit
- **rekonstruierte Daten:** Spuren in den Spur-Detektoren, Blöcke oder Cluster in den Calorimetern, rekonstruierte Zerfallsvertizes, Erfordert immer noch komplexe Tree Struktur

- **Summary Daten:** rekonstruierte abstrakte Objekte: Jets, Leptonen, Photonen, Missing Momentum Vektor, ...
- **Globale Summary:** Zahl der Spuren oder Jets, Energie in Kalorimeter, ... Für einfache Charakterisierung genügt flaches Layout, feste Zahl von Parametern pro Ereignis.

End-Analysen verwenden i.d.R. letztere Formate...

3.2 Trees in Root

```

TFile *f = new TFile("/project/etp/gduckeck/Z0-Versuch/root/ntz0mhmc.root"); // local file 1
TFile *f = TFile::Open("http://www.etp.physik.uni-muenchen.de/kurs/comp10/uebungen/Z0-Versuch/data/"); // remote file 2
f->ls(); // liefert Info zu File Struktur 3
// KEY: TTree h5000;1 EVENT // h5000 ist Name des Trees 4
h5000->Print(); // liefert Info zu Variablen im Tree 5
h5000->Draw("Ncharged"); // fuellt 1D Histogramm mit Variable Ncharged (automatische Histo Erzeugung) 6
h5000->Draw("N_ecal:Ncharged"); // fuellt 2D Histogramm mit Variable Ncharged vs N_ecal 7
TH1F* h1 = new TH1F("nc","N Tracks", 50, 0, 50); // Buche 1d Histo 8
h5000->Draw("Ncharged>>nc"); // fuellt 1D Histogramm mit Variable Ncharged in gebuchtes Histo 9
h5000->Draw("Ncharged","E_ecal>10"); // fuellt 1D Histogramm mit Variable Ncharged wenn Cut erfuehlt ist 10

```

3.3 Grafische Tree-Browser

Verschiedene Varianten in Root für GUI basierte File bzw. Tree Browser:

- Starten Sie einen `TBrowser` mit:

```
TBrowser b
```

- Klicken Sie im linken Unterfenster auf `ROOT Files` und anschliessen im rechten Unterfenster auf die entsprechende ROOT ntuple file
- Navigieren Sie durch die Datei durch Doppel-Klicken auf die jeweiligen Elemente
- Weitere Features im TTreeView:
 - Start direkt von Kommandozeile:
`TTreeView tv("h5000")`
 - oder via rechte Maustaste im `TBrowser->StartViewer`
- Oder noch besser über HistPresent

Direktes *Tree-Browsing* ok für einmalige, schnelle Checks.

Für systematische, reproduzierbare Analysen besser C++ Klassen für Analysen verwenden ⇒ später mehr

3.4 ASCII-Datei in ROOT-Tree einlesen

Mit der Methode `TTree::ReadFile` können ASCII Daten automatisch in eine TTree-Struktur eingelesen und abgespeichert werden:

`Long64_t TTree::ReadFile(const char *filename, const char *branchDescriptor)` erzeugt oder liest 'branches' aus einer Dateien mit dem Namen 'filename'.

Ein Beispiel ist gegeben in folgender Datei: [readfile.C](#)

4 Datenanalyse – ausführliches Beispiel Z0-Versuch

4.1 Einführung

Der Z0 versuch aus dem F-Praktikum gibt gutes Beispiel für Datenanalyse in Teilchenphysik.

- Verwendet Daten des OPAL Experiments am e^+e^- Beschleuniger LEP (Vorläufer von LHC).
- LEP Daten vergleichsweise sauber und “einfach” zu interpretieren.
- Aufgabe: Bestimmung des Wirkungsquerschnitts für die Reaktionen $e^+e^- \rightarrow Z^0 \rightarrow q\bar{q}$ und $e^+e^- \rightarrow Z^0 \rightarrow \mu^+\mu^-$
- Datensätze in </project/etp/gduckeck/Z0-Versuch/root> bzw. <http://www.etp.physik.uni-muenchen.de/kurs/comp10/uebungen/Z0-Versuch/data/>
- Vorgehen:
 - Erstellen geeigneter Algorithmen (=Schnitte) zur Selektion dieser Endzustände

Simulierte Daten für Signal und Untergrund Reaktionen :

`ntz0mhmc.root` : $e^+e^- \rightarrow Z^0 \rightarrow q\bar{q}$

`ntz0mmmc.root` : $e^+e^- \rightarrow Z^0 \rightarrow \mu^+\mu^-$

`ntz0eemc.root` : $e^+e^- \rightarrow Z^0 \rightarrow e^+e^-$

`ntz0ttmc.root` : $e^+e^- \rightarrow Z^0 \rightarrow \tau^+\tau^-$

- * Optimierung der Schnitte \Rightarrow hohe Signal-Effizienz, wenig Untergrund
 - * Bestimmung von entsprechenden Korrekturfaktoren
 - Selektion anwenden auf echte Datensätze (7 verschiedene Schwerpunktsenergien)
`ntz0e1.root - ntz0e7.root`
 - Korrektur und Berechnung der Wirkungsquerschnitte
 - Vergleich mit theoretischen Vorhersagen und weitere Interpretation
- Ausführliche Dokumentation in
<http://www.etp.physik.uni-muenchen.de/fp-versuch/index.html>.

4.2 Ausarbeitung der Selektion

Zur Ausarbeitung der Selektionskriterien empfiehlt sich folgendes Vorgehen:

- Überlegen welche Grössen in Frage kommen für Cuts
- Verteilungen dieser Grössen studieren für Signal und Untergrund Daten
- Satz von Schnitten erstellen
- Vergleichen der Verteilungen Daten – Monte Carlo
 evt weitere Cuts nötig

Praktisches Beispiel für Vorgehen (Verwendung von Hilfsklasse **GDUtils.C**).

```
{ 1
// some example commands for Z0-Versuch ntuple analysis 2
.L GDUtils.C+ ; // Load utility Macro 3
GDUtils u; // create object 4
// TFile *f = new TFile("/project/etp/gduckeck/Z0-Versuch/root/ntz0mhmc.root"); 5
TFile *f1 = TFile::Open("http://www.etp.physik.uni-muenchen.de/kurs/comp10/uebungen/Z0-Versuch/da 6
h1 = u.H1("e-mc","E_Ecal", 100, 0, 100); // book and fill histo from tree 7
h5000->Draw("E_ecal>>e-mc"); 8
```

```
// TFile *f2 = new TFile("/project/etp/gduckeck/Z0-Versuch/root/ntz0e4.root");          9
TFile *f2 = TFile::Open("http://www.etp.physik.uni-muenchen.de/kurs/comp10/uebungen/Z0-Versuch/da
h2 = u.H1("e-d","E_Ecal", 100, 0, 100); // book and fill histo from tree          11
h5000->Draw("E_ecal>>e-d");                                                    12
u.NewCanvas(); // canvas with 2 pads                                           13
u.Zone(1,2);                                                                    14
u.Drawh1("e-d"); // draw histos                                               15
u.Drawh1("e-mc");                                                                16
u.Over("e-d","e-mc"); // overlay histos with normalization                   17
u.Over("e-d","e-mc", 0, 20, 70);                                              18
}                                                                                19
```

GDUtils Klasse enthält einige nützliche Funktionen um Zugriff auf Histogramme und Zeichnen zu vereinfachen.

4.3 Selektion in C++ mit eigener TSelector Klasse

Für eine komplexe, reproduzierbare Selektion empfiehlt es sich die Schnitte in einer richtigen C++ Klasse bzw. Funktion zu definieren und damit den Tree zu prozessieren.

Eine Möglichkeit ist mit `TreeName->MakeClass("MyClass");` so eine Klasse mit Root zu erzeugen.

Eine zweite ähnliche Variante ist die Erzeugung einer *TSelector* Klasse:

- `h5000->MakeSelector("MyZ0Selector");`
Vorher Tree File öffnen (`TFile f("ntz0mhmc.root")`)
- Generiert Dateien `MyZ0Selector.h` und `MyZ0Selector.C` mit Klasse `MyZ0Selector`.
- **MyZ0Selector** ist abgeleitete Klasse von **TSelector**
- Grundlegende Methoden:
 - `Begin()` zur Initialisierung (Histogramme anlegen, etc.)
 - `Terminate()` zum Abschliessen (Ergebnisse und Histogramme bzw abspeichern, etc.)
 - `Process(entry)` zum Prozessieren, wird für jedes Event/entry gerufen, Schnitte anwenden, Histogramme füllen, etc.)
 - `Process()` Methode sollte Aufruf von `GetEntry(entry);` enthalten um erstmal komplettes Event einzulesen.

Zum Prozessieren des Trees am Besten:

```
.L MyZ0Selector.C+ // Klasse Kompilieren/Laden
MyZ0Selector *s1 = new MyZ0Selector() // Objekt anlegen
TFile f("ntz0e4.root") // Tree File oeffnen
h5000->Process(s1) // Tree Prozessieren
```

TSelector/MakeSelector ist modernere Variante von **MakeClass/MyClass**.

⇒ flexibler, kann auch mit **PROOF** (Paralleles Prozessieren mit Root) verwendet werden.

Weiteres Vorgehen für Z0 Versuch:

- Getrennte TSelector Klassen für jeweils Hadron- und Muon-Selektion erstellen
- Zum Prozessieren der verschiedenen Datensätze jeweils eigenes Objekt anlegen:

```
MyHadSelector *shd1 = new MyHadSelector() // Objekt anlegen
TFile f("ntz0e1.root") // Tree File oeffnen
h5000->Process(shd1) // Tree Prozessieren
...
MyHadSelector *shd2 = new MyHadSelector() // Objekt anlegen
TFile f("ntz0e2.root") // Tree File oeffnen
h5000->Process(shd2) // Tree Prozessieren
...
```
- Objekte enthalten anschliessend Histogramme und sonstige Ergebnisse der Selektionen.

Beispiele:

SomeZ0Selector.h

SomeZ0Selector.C

runZ0sel.C

Beispiel für Korrektur bzw Wirkungsquerschnitt-Berechnung: [calcXS.C](#)

Beispiel-Skript zum Überlagern von Histogrammen: [stackHist.C](#)

5 HistPresent

HistPresent - Easy interactive analysis within the ROOT framework

O. Schaile
University of Munich, D-85748 Garching, Germany
email: Otto.Schaile@LMU.de

Presented at the ROOT2005 WORKSHOP, CERN, 26.Sept.05

5.1 Introduction

HistPresent was developed within a project for online and offline data analysis, MARaBOU, to provide access to ROOT objects like histograms, graphs and ntuples as typically used in accelerator experiments. Special emphasis is placed on the simplicity of the user interface to enable unexperienced users to display histograms and do simple operations like zooming and signal/background fitting. At the same time the full functionality of ROOT is made available like e.g. fitting of slices with complicated functions by providing templates for macros. Some extra features like simultaneous projections along X and Y and rotations of 2-dim histograms are built in.

HistPresent also exploits the interactive graphics capabilities of ROOT. In addition to ROOT's built-in graphics primitives like graphs, paves, curlylines etc. it allows interactive insertion of histograms, images, functions, xsplines and text files containing LaTeX code. Features for editing common objects on a grid are provided. Compound objects may be defined, extracted, stored in a file and reinserted with new scale and angle.

5.2 The MARaBOU project

MARaBOU is a framework for online data acquisition and as offline analysis for nuclear physics experiments. It was developed at the Tandem Laboratory in Garching operated by the University (LMU) and Technical University (TUM) Munich. Design principle of MARaBOU is that the user only sees and communicates with a ROOT based system running on Linux whereas the hardware of the experiment is housed in CAMAC and VME controlled by processors running Lynx OS on VME processors.

Fig. 1 gives an overview on the dataflow within the system.

5.3 Display histogram

Fig. 2 shows HistPresent's startup window, a file list, a list of a file contents and a widget displayed for a NTuple. All these widgets are implemented using `TButton` objects. Pressing the buttons triggers the obvious actions like showing the content of a ROOT file or displaying a histogram. The `grey` button left of a name is used as a selection. It turns green if it is active. This is used in

operations like `Show all selected hists` or with the arithmetics operators. E.g. Selecting `hpx` and `hpy2` then pressing `+` would addup histograms `hpx` and `hpy2` resulting in a histogram `hpx_plus_hpy2` which is displayed and added to the objects in memory.

Showing values of a Tree can be done in several ways:

- Just pressing a name fills and shows a 1-dim histogram
- Selecting 2 or 3 names and pressing `Show leaves` fills and shows a 2 or 3-dim histogram
- With `Edit expression` any legal expression may be formed. The input widget used for that implements a history (see fig. 2) and text completion (tab key) for variable names and frequently used functions like `TMath::Sqrt`

A histogram is displayed in a special canvas inheriting from `TCanvas` with extra popup menus providing actions like `Expand`, `Superimpose`, `Projections` etc. Note: Up to now all operations only require the left mouse button whereas the standard ROOT actions from context dependent popup menus triggered by the right mouse button are still available.

The following picture shows examples: `Show selected hists` in same range, and a 2-dim histogram with a graphical cut (`TCutG`) applied projected on both axes.

5.4 Fitting functions

Among others HistPresent's canvas provides a popup menu `Fit/Calibrate` (see fig. 3). Several predefined functions like `Polynomials` and `Gauss+High Tail` are available. In this case marks which are activated with the middle mouse button are used to define limits and approximate means for a fit. Start parameters are estimated and editable by the user.

A second more general way is to use a ROOT macro. Commands `Edit User Fit Macro` and `Execute User Fit Macro` are provided for this. If with `Edit User Fit Macro` a non existing file name is given the user can choose from a few examples as templates. Again this is to put the entrance threshold to use ROOT as low as possible.

For 2-dim histograms hooks are provided to fit slices and 2d functions. Also here examples are provided to start with.

Fig. 5 shows the 2-dim histogram from the builtin tutorial and the result of the example fit macro.

5.5 The graphics editor

A graphics editor canvas is invoked from the `File` popup in the startup window. For convenience three sizes are predefined: A4-portrait, A4-landscape and user. In the canvas an edit grid is provided. This is implemented by overloading `TCanvas::HandleInput` by forcing mouse click coordinates on the user defined grid. To overcome the problem of the finite pixel resolution a method is provided to align all objects (selectable by type) in the canvas at the grid. Global editing facilities (e.g. change color of all `CurlyLines`, `Delete enclosed objects` and zooming are available.

In addition to ROOT's builtin graphics primitives like `TArcs`, `TCurlyLines` the following features / objects are accessible by a menu (see fig. 7):

- `Insert images (TASImage), histograms, graphs`. The user has to define a `Pad` as container for the object.
- `Insert a function`. After giving the number of parameters a widget (fig. 6) allows input of the needed values.
- `Insert Text / Latex from file or keyboard`. This widget facilitates input of `Text / LaTeX` by implementing input from and providing a history facility for input from keyboard. Text attributes may be adjusted from the widget (see fig. 6). Optionally a filter which tries to convert standard LaTeX to ROOT's Latex may be switched on.
- `Insert compound object`. Compound objects may be defined by enclosing graphic elements by a `TCutG` and saved on file. They can be reinserted applying a scale and rotation.

- `Draw a XSpline`. XSplines as used e.g. in XFig are very simple to control by moving the controlpoints or changing the shapefactors between -1 and 1. HistPresent provides a Control Graph Mixer for that. Parallel graphs, optionally with railway like filling, may be added.

5.6 Documentation, Availability, Download

The documentation of HistPresent can be found here:

<http://www.bl.physik.uni-muenchen.de/marabou/html/hpr>

HistPresent is part of MARaBOUs CVS repository. The structure of directories and the Makefile is similiar to the ones of ROOT itself. It was only used and tested on Linux type systems.

To install HistPresent please consult this URL:

<http://www.bl.physik.uni-muenchen.de/marabou/html/UpdateHprFromCVS.html>

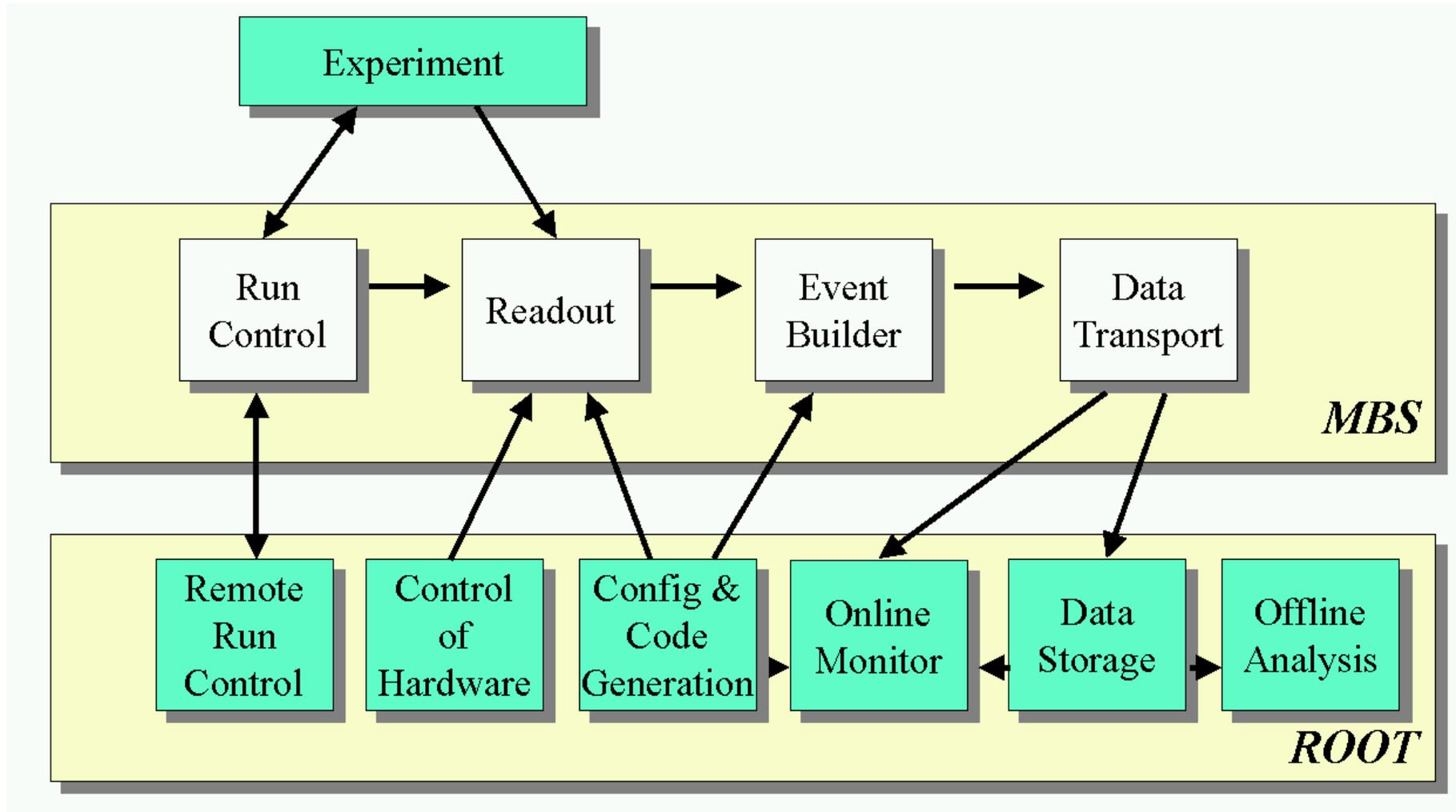


Figure 1: Marabou Tasks

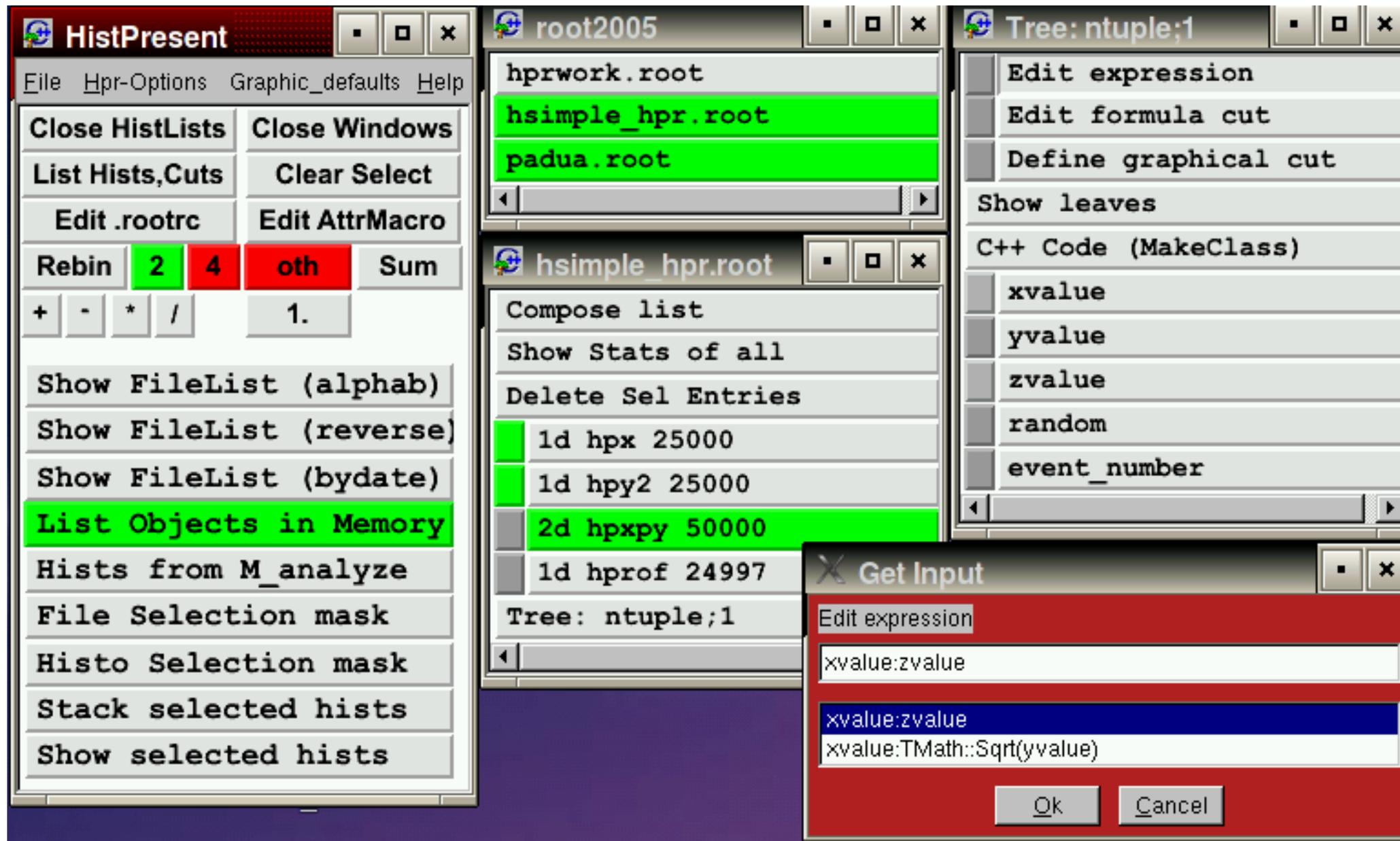


Figure 2: A sample session

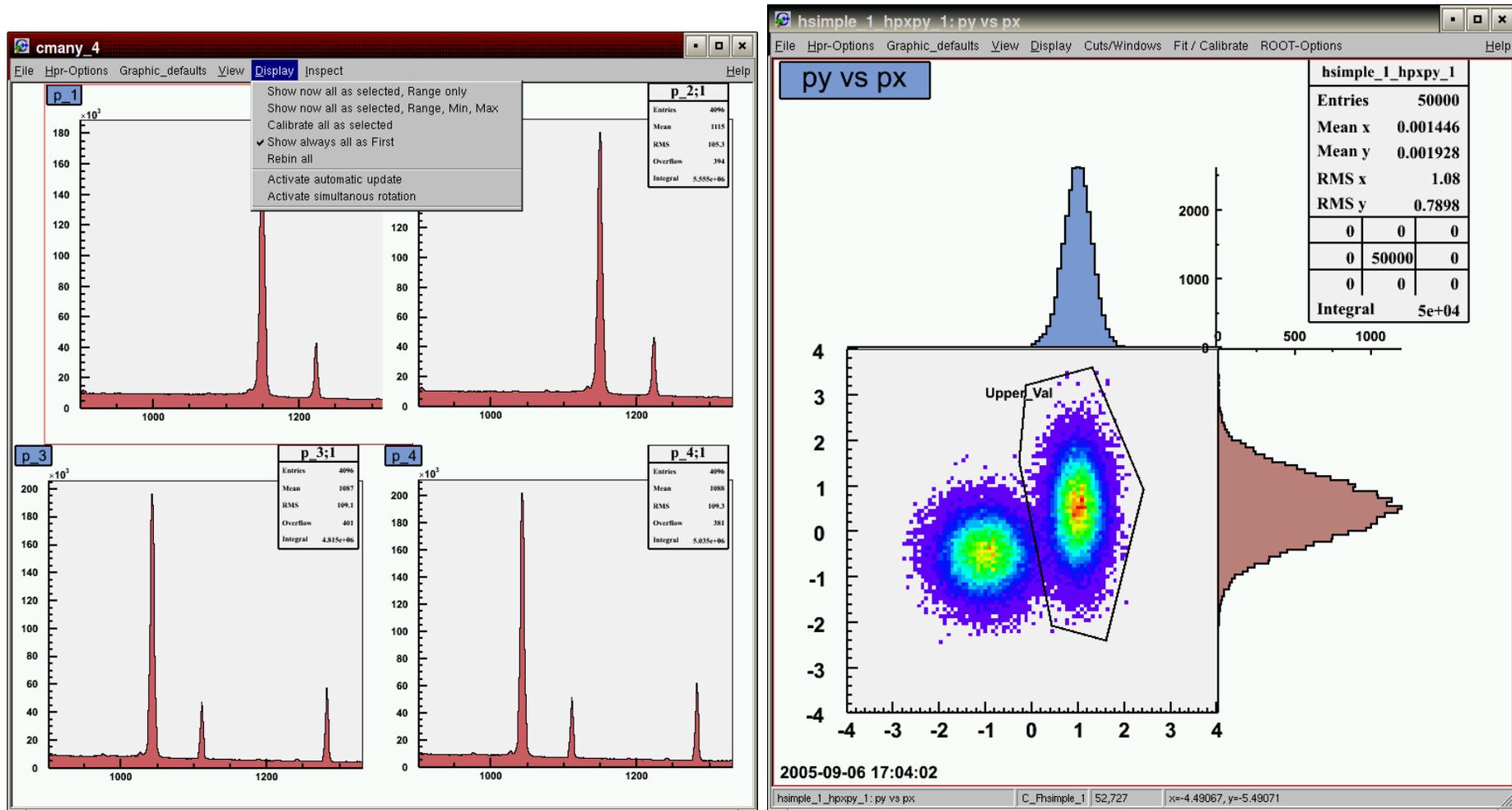


Figure 3: Show selected hists, Display_both option with graphical cut

The screenshot displays a software interface for histogram fitting. The main window, titled "hsimple_hpr_hpx_1: This is the px distribution", shows a histogram with a peak at 800. A "Fit / Calibrate" menu is open, listing various fitting options. A "Template Fit Macros" dialog is open, showing options for fitting with two Gaussians, an exponential plus Gaussian, Breit Wigner, three Landaus, and a polynomial plus sine. A "Start parameters" dialog is also open, displaying a table of parameters for background and Gaussian components.

Start parameters

	StartVal	Low Lim	Up Lim	fix it	bound
Bg_Const	0	0	0	<input type="checkbox"/>	<input type="checkbox"/>
Bg_Slope	0	0	0	<input type="checkbox"/>	<input type="checkbox"/>
Ga_Width	1.98	0	0	<input type="checkbox"/>	<input type="checkbox"/>
Ga_Cont	2.5e+04	0	0	<input type="checkbox"/>	<input type="checkbox"/>
Ga_Mean	0	0	0	<input type="checkbox"/>	<input type="checkbox"/>

Figure 4: Widgets used with Fitting

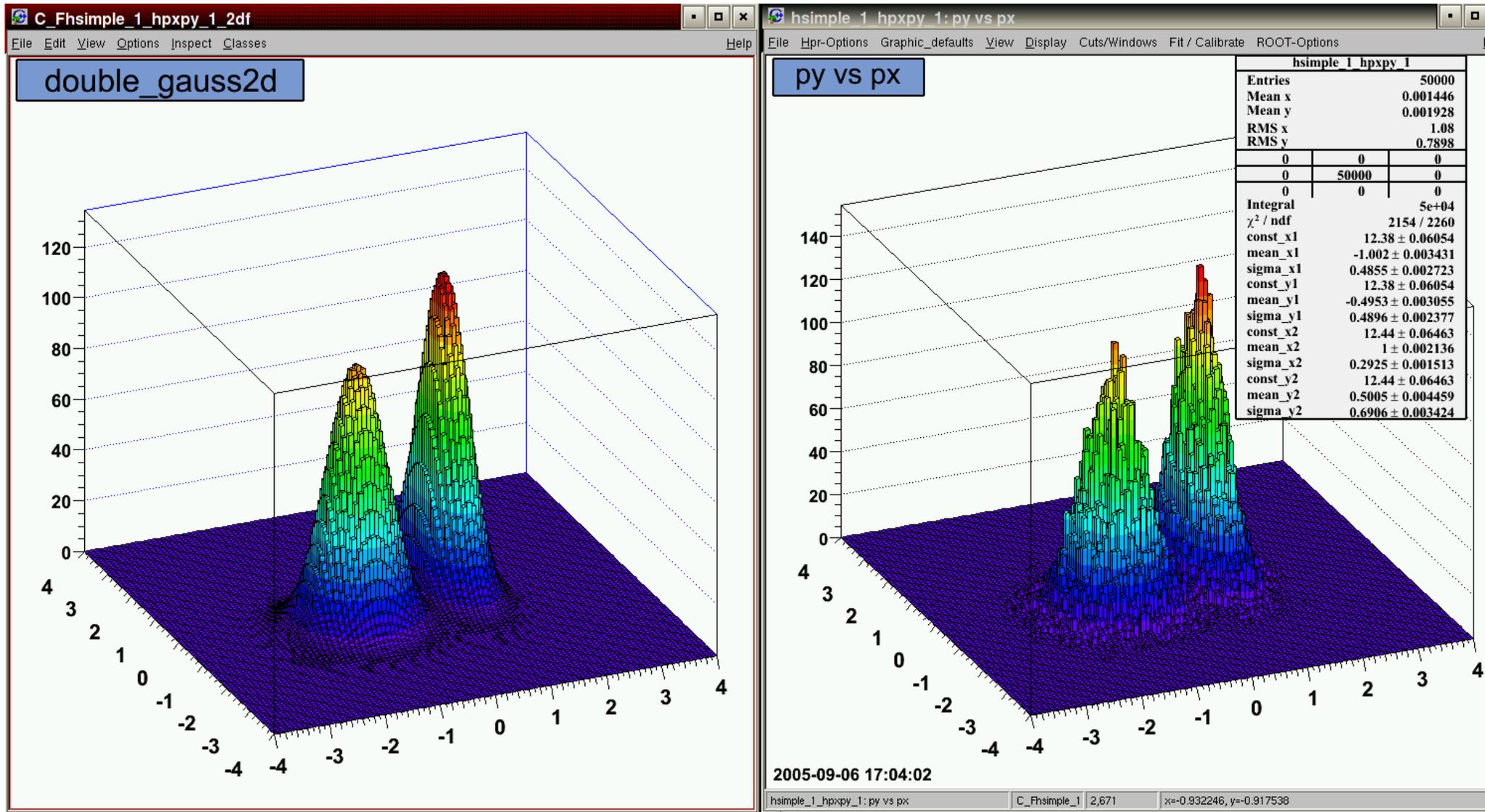


Figure 5: Example of 2d fit

Get Input:Text (latex) string

X Position	0
Y Position	0
Line spacing	10
Text size	0.02
Text font	6. helvetica bold
Text precision	2
Text color	■
Text alignment	■
Text angle	0
Mark as compound	<input type="checkbox"/>
Apply latex filter	<input checked="" type="checkbox"/>

Text (latex) string

s misst die Gesamtenergie E_{cm}^2/c^2 im Schwerpu

$p^2 = E^2/c^2 - (\vec{p})^2 = M^2 c^2$
 $ab = a_0 b_0 - a_1 b_1 - a_2 b_2 - a_3 b_3$
s misst die Gesamtenergie E_{cm}^2/c^2 im Schwerpu
 $s := (p_a + p_b)^2$

Get Input:Function formula

Function Name	fun2
X axis title	x
Y axis title	y
Lower limit (from)	0
Upper limit (to)	10
Value of Parameter 0	1
Value of Parameter 1	-0.2
Value of Parameter 2	4
Drawing color	■
Opacity of pad (0-100)	30
Use new(selected) pad	<input type="radio"/>
Use same (selected) pad	<input checked="" type="radio"/>
Create separate canvas	<input type="radio"/>

Function formula

$[0] \cdot \exp([1] \cdot x) \cdot \cos([2] \cdot x)$

$[0] \cdot \exp([1] \cdot x) \cdot \cos([2] \cdot x)$
 $[0] \cdot \exp([1] \cdot x \cdot x) \cdot \sin([2] \cdot x)$

Figure 6: Input Latex, Input function

page_1

File Hpr-Options Graphic_defaults View Hpr-Edit Inspect

Style

Name
::TLatex

Line
2

Text
23
6. helvetica bold
11 Bottom, Left

Hpr Edit Co

Insert image (gif, jpg)

Insert histogram

Insert graph

Insert Function

Text (Latex) from file

Text (Latex) from keyboard

$$\frac{2s}{\pi\alpha^2 d\cos\theta} (e^+e^- \rightarrow f\bar{f}) = \left| \frac{1}{1-\Delta\alpha} \right|^2 (1 + \cos^2\theta) + 4 \operatorname{Re} \left\{ \frac{2}{1-\Delta\alpha} \chi(s) \left[\tilde{g}_v^e \tilde{g}_v^f (1 + \cos^2\theta) + 2 \tilde{g}_a^e \tilde{g}_a^f \cos\theta \right] \right\} + 16 |\chi(s)|^2 \left[(\tilde{g}_a^{e2} + \tilde{g}_v^{e2})(\tilde{g}_a^{f2} + \tilde{g}_v^{f2})(1 + \cos^2\theta) + 8 \tilde{g}_a^e \tilde{g}_a^f \tilde{g}_v^e \tilde{g}_v^f \cos\theta \right]$$

April 2014

6 Eventdisplay

Einzelne Datenereignisse in einem Teilchenphysikexperiment werden mit einem sog. „Eventdisplay“ visualisiert. Dabei werden Spuren und Treffer der Reaktionsprodukte einer Teilchenkollision in den einzelnen Unterdetektoren graphisch dargestellt.

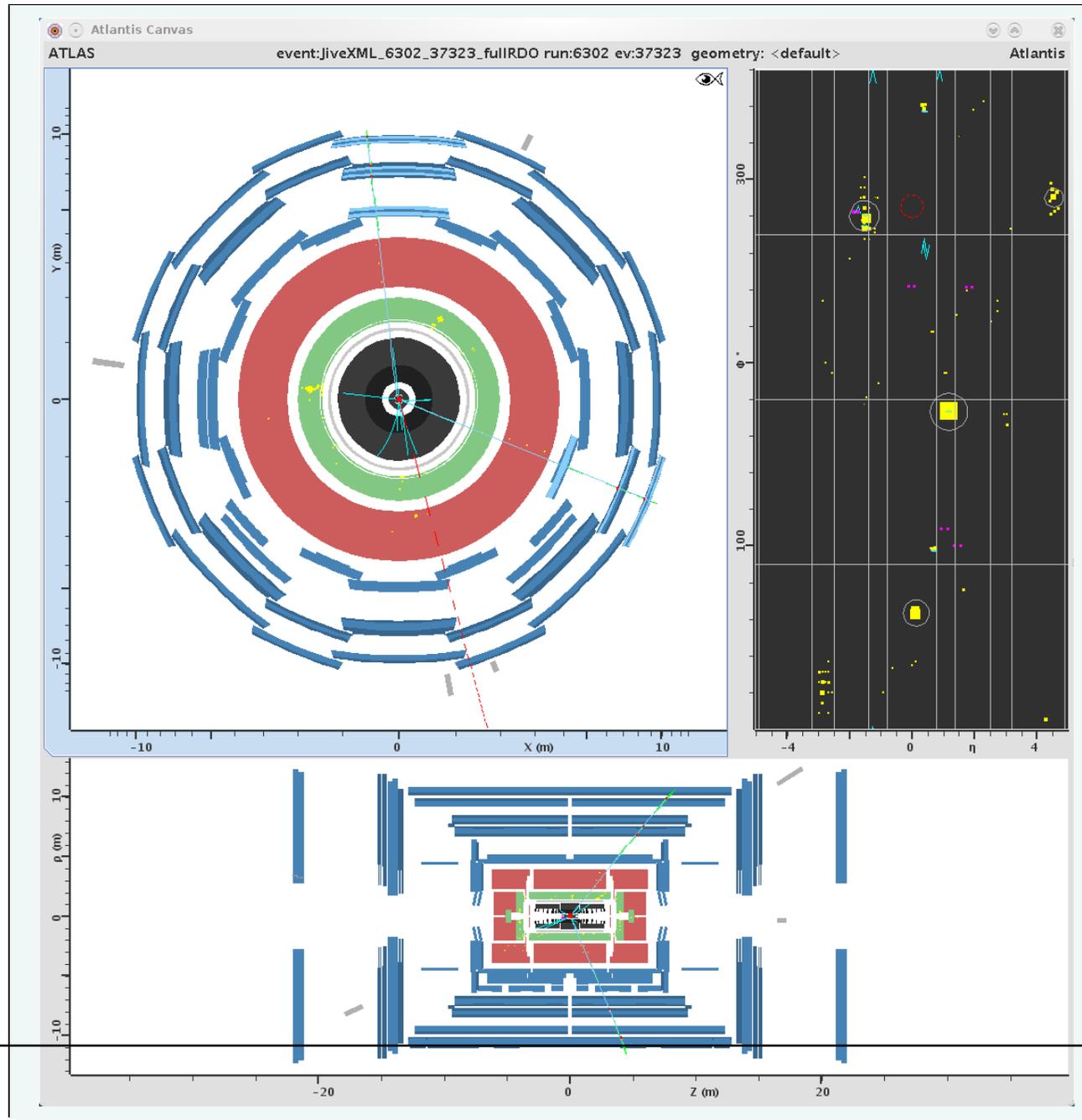
6.1 MasterClasses, Minerva

Ein interaktives Web-Tutorial zum ATLAS Event-Display gibt es unter:

- [ATLAS MasterClasses](#)
- [Minerva Wiki](#)

Minvera Installation und Start:

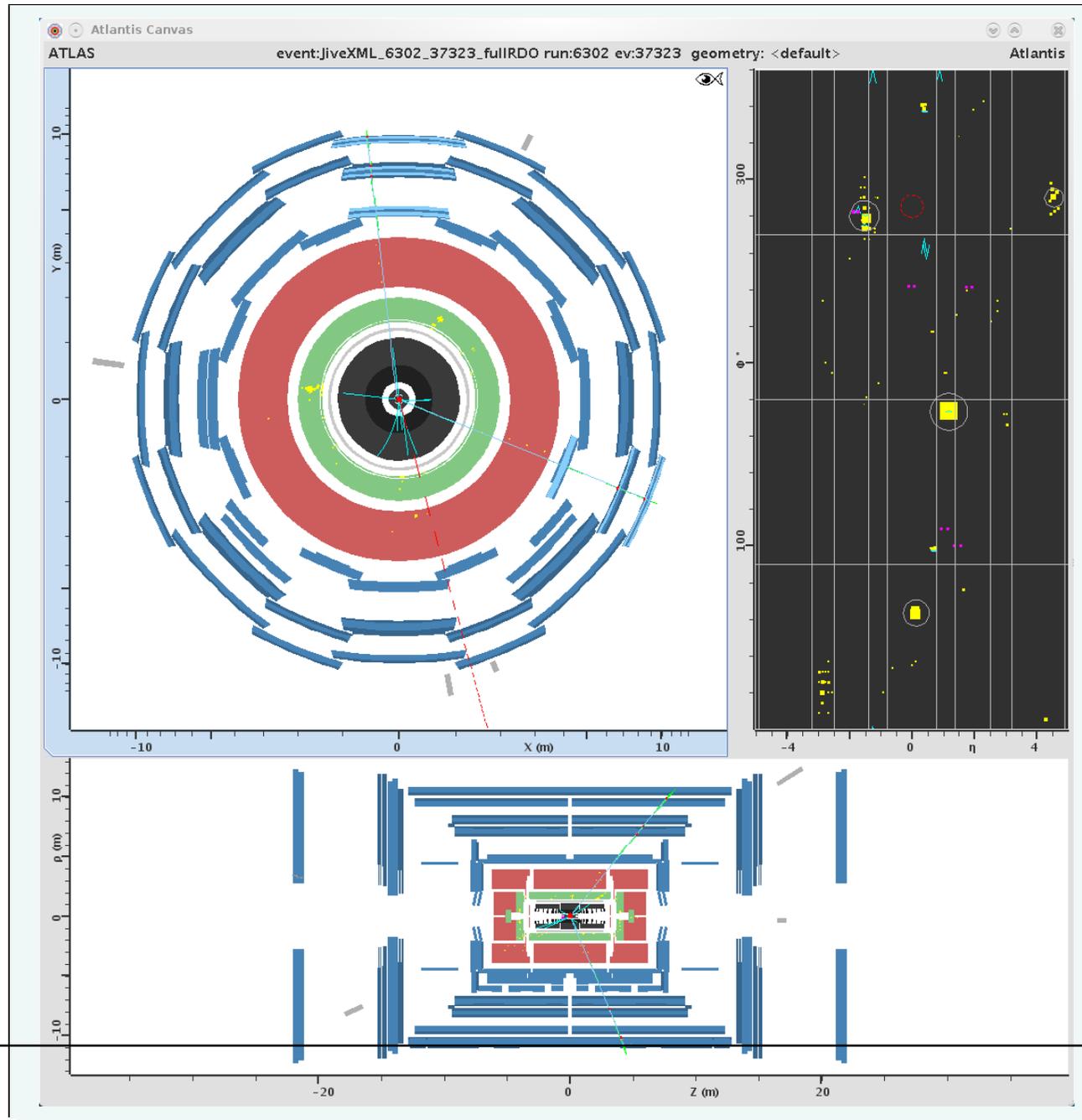
```
mkdir minerva
cd minerva
wget http://epweb2.ph.bham.ac.uk/user/mclaughlan/minerva/MINERVA.zip
unzip MINERVA.zip
java -jar atlantis.jar
```



6.2 ATLANTIS

Atlantis ist ein „Eventdisplay“ für das ATLAS Experiment am Large Hadron Collider (LHC) am CERN in Genf. Die primäre Aufgabe des Programms ist die graphische Untersuchung und das Verstehen der Physik vollständiger Ereignisse. Weiterhin können mit Hilfe von ATLANTIS Rekonstruktions- und Analysealgorithmen entwickelt werden und Unterdetektoren besser verstanden werden. Auch ist es ein Werkzeug, um Bilder und Animationen für Publikationen, Präsentationen und Ausstellungen zu erzeugen.

Folgendes Bild zeigt ein Ereignis aus verschiedenen Richtungen:



6.3 Setup und Start von ATLANTIS

- In einem Terminalfenster führen Sie folgende Befehle aus:

```
ssh etpopt01
export ATLAS_LOCAL_ROOT_BASE=/cvmfs/atlas.cern.ch/repo/ATLASLocalRootBase
source $ATLAS_LOCAL_ROOT_BASE/user/atlasLocalSetup.sh
```

- Um die ATLAS Softwareumgebung zu starten, führen Sie folgenden Befehl aus:

```
asetup 17.2.0
```

wobei 17.2.0 eine spezielle ATLAS Softwareversion ist.

- Das ATLANTIS Eventdisplay startet man anschließend mit:

```
atlantis
```

- Machen Sie sich mit ATLANTIS vertraut und versuchen Sie mehrere Ereignisse aus verschiedenen Perspektiven und Auflösungen darzustellen.

6.4 Übungen mit ATLANTIS

In dem Verzeichnis [/project/etp/rootcourse/data/xml/](#) gibt es mehrere Archivedateien

`eventsX.zip` mit $X=[1,5]$. Diese Ereignisse in einer Archivdatei stammen jeweils von simulierten Daten folgender Prozesse:

- $Z \rightarrow \mu^+ \mu^-$
- $Z \rightarrow e^+ e^-$
- $t\bar{t}$
- SUSY SU4
- $H \rightarrow ZZ \rightarrow 4 \text{ Leptonen}$

Versuchen Sie herauszufinden welche Prozesse zur jeweiligen Archivdatei gehören.

Einzelne Myon-Ereignisse die Abschnitt zur Datenselektion verwendet werden, gibt es als Archivdatei unter:

`/project/etp/rootcourse/data/xml/events.zip` Laden Sie diese Datei im 'Files' Menü und betrachten Sie die besonderen Eigenschaften.

6.5 Weiterführendes Material

- **ATLANTIS Homepage** mit JAVA webstart application
- **ATLANTIS Wiki**
- **Eine kurze Einführung** (leider nur ATLAS intern)

7 Datenanalyse

7.1 Einführung

ROOT ermöglicht die effiziente und schnelle Analyse von sehr großen Datenmengen. Daten können entweder in ASCII-Format gespeichert und mit normalen C/C++ Befehlen eingelesen und weiterverarbeitet werden. Sehr viel effizienter ist die Speicherung im ROOT-tuple- bzw. n-tuple-Format. In diesem Format werden Daten und ihre Eigenschaften bzw. Variablen in sog. Baum-Format (Trees) strukturiert.

7.2 TBrowser

Machen Sie sich mit einem ntuple vertraut:

- Starten Sie ROOT mit:

```
root /data/etp/rootcourse/data/user08.Marie-HeleneGenest.ganga.McAtNloZmumu_1Lep
```

- Starten Sie einen TBrowser mit:

```
TBrowser b
```

- Klicken Sie im linken Unterfenster auf `ROOT Files` und anschliessen im rechten Unterfenster auf die entsprechende ROOT ntuple file

- Navigieren Sie durch die Datei durch Doppel-Klicken auf die jeweiligen Elemente

7.3 MakeClass

Generieren Sie ein Programmgerüst, um mit dem ROOT-tuple effizient arbeiten zu können.

- Starten Sie ROOT:

```
root
```

- Öffnen Sie eine Datei mit (Achten Sie auf die richtigen Anführungszeichen):

```
TFile f("/data/etp/rootcourse/data/user08.Marie-HeleneGenest.ganga.McAtNloZmumu
```

- Erzeugen Sie ein Programmgerüst aus dem vorgebenen ROOT Tree `ControlSample0` mit:

```
ControlSample0->MakeClass("MyClass");
```

- Es existieren nun die Dateien `MyClass.h` und `MyClass.C` , die Sie mit einem Texteditor öffnen sollten.
- Fügen Sie in `MyClass.h` um Zeile 14 folgende Zeile ein:

```
#include <vector>
```

- Führen Sie folgendes Kommando aus, um die Klasse `MyClass` und die zugehörigen Methoden in den ROOT Cint Speicher zu laden:

```
.L MyClass.C
```

Syntaxfehler im C/C++ Code können sehr leicht zu Programmabstürzen führen.

- Folgendes Kommando ruft vor dem Laden zusätzlich den ACLiC Compiler auf und erzeugt eine shared library:

```
.L MyClass.C++
```

Somit können Syntax-Fehler effizient abgefangen werden.

7.4 **MyClass verwenden**

Die Instanz von `MyClass` kann nun dazu verwendet werden, die im ROOT-tuple gespeicherten Ereignisse zu prozessieren. Dazu existiert die Methode `MyClass::Loop()`

- Fügen Sie in `MyClass.C` um Zeile 7 folgende Zeile ein:

```
#include <iostream>
```

- In die `for`-Schleife um Zeile 46 fügen Sie ein:

```
std::cout << jentry << std::endl;
```

- Führen Sie nun folgende Befehle aus:

```
.L MyClass.C++
```

```
MyClass t
```

```
t.Loop()
```

Es sollten nun zeilenweise Zahlen auf dem Bildschirm ausgegeben werden.

7.5 Ein Histogramm füllen

- Erzeugen Sie ein Histogramm, indem Sie folgende Zeile vor die `for`-Schleife einfügen:

```
TH1D *h_n_mu = new TH1D("nMuons", "Number of Muons per event",  
10, -0.5, 9.5);
```

- Füllen Sie dieses Histogramm, indem Sie folgende Zeile in die `for`-Schleife einfügen:

```
h_n_mu->Fill(Mu_N);
```

Die Variable `Mu_N` ist eine der zahlreichen Variablen im `root-tuple`, sichtbar entweder mit dem `TBrowser` oder in `MyClass.h`.

- Erzeugen Sie einen `Canvas` zur Anzeige des Histogramms, indem Sie folgende Zeilen nach der

```
for-Schleife einfügen: TStyle *style = gROOT->GetStyle("Plain");  
gROOT->SetStyle("Plain");
```

```
TCanvas *can1=new TCanvas("can1", "Muons", 0, 0, 800, 800);  
can1->cd(1);  
h_n_mu->Draw();
```

- Übersetzen und laden Sie den neuen code mit:

```
.L MyClass.C++  
MyClass t  
t.Loop()
```

7.6 Eine Selektion starten

Normalerweise möchte man nicht alle Ereignisse eines ROOT tuples anzeigen, sondern nur bestimmte Ereignisse mit speziellen Eigenschaften anzeigen. Folgende Programmzeilen füllen einen `TLorentzVector` mit Spurinformatoren von Myonen.

- Fügen Sie folgende Zeilen am Anfang von `MyClass.C` ein:

```
#include <TLorentzVector.h>
#include <TMath.h>
```

- Definieren Sie mehrere Histogramme, `TLorentzVector` und temporäre Variablen mit folgenden Zeilen vor der `for`-Schleife:

```
TH1D *h_pt_mu1 = new TH1D("ptMuon1", "pt of Muon 1", 50, 0., 100.*1000.);
TH1D *h_pt_mu2 = new TH1D("ptMuon2", "pt of Muon 2", 50, 0., 100.*1000.);
TLorentzVector m1(0, 0, 0, 0), m2(0, 0, 0, 0), m(0, 0, 0, 0);
```

- Fügen Sie folgende Zeilen in der `for`-Schleife ein:

```
m1.SetPxPyPzE(0, 0, 0, 0);
m2.SetPxPyPzE(0, 0, 0, 0);
m.SetPxPyPzE(0, 0, 0, 0);

for (int i=0; i<Mu_N;i++) {
m.SetPxPyPzE((*Mu_px)[i], (*Mu_py)[i], (*Mu_pz)[i], (*Mu_E)[i]);

if (m.Pt()>m1.Pt()) {
```

```
m2 = m1;
m1 = m;
}
else if((m.Pt()>m2.Pt()) && (m.Pt() < m1.Pt())) {
m2 = m;
}
}

if ((m1.Pt()>5.*1000.) &&
(m2.Pt()>5.*1000.)) {
h_pt_mu1->Fill(m1.Pt());
h_pt_mu2->Fill(m2.Pt());
}
```

Diese Zeilen füllen die Histogramme mit den Myon Transversalimpulsen p_T und selektieren nur Ereignisse mit zwei Myonen und $p_T > 5$ GeV.

- Ein Canvas kann auf in mehrer TPad unterteilt werden, so daß mehrere Histogramme gleichzeitig angezeigt werden können: `TStyle *style = gROOT->GetStyle("Plain");`
`gROOT->SetStyle("Plain");`

```
TCanvas *can1=new TCanvas("can1","Muons",0,0,800,800);
can1->Divide(2,2);
can1->cd(1);
h_n_mu->Draw();
can1->cd(2);
h_pt_mu1->Draw();
can1->cd(3);
h_pt_mu2->Draw();
```

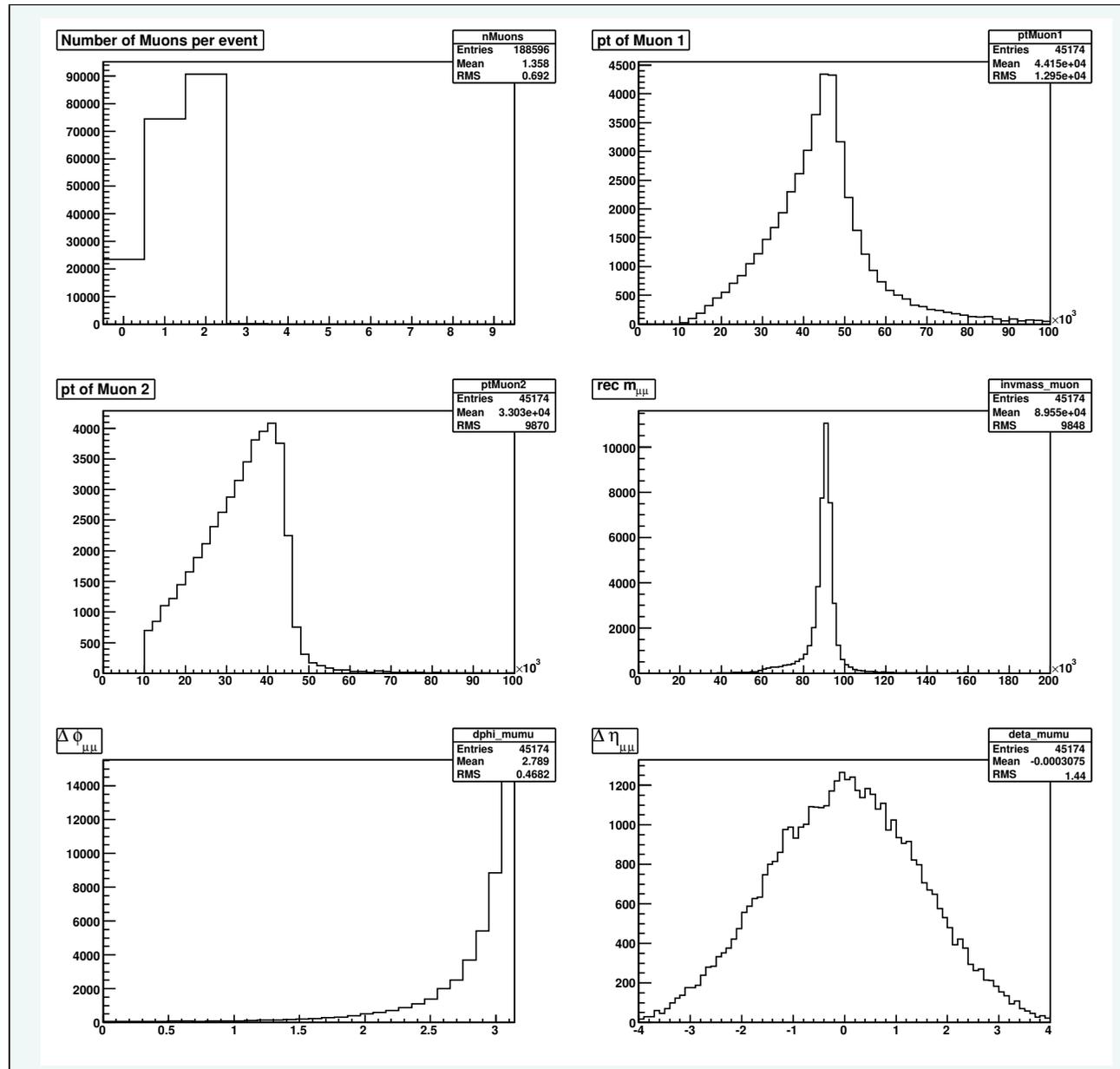
7.7 Eine erweiterte Selektion

Erweitern Sie nun den schon entstandenen code um folgende Histogramme:

- Die invariante Masse der beiden Myonen mit höchstem Transversalimpuls. Verwenden Sie für die Berechnung die Methode `TLorentzVector.M()` und alternativ den relativistischen Energie-Impulsatz.
- Die Differenz $\Delta\phi$ im Azimutalwinkel der beiden Myonen. Verwenden Sie `Mu_phi`.
- Die Differenz der Pseudorapidität $\Delta\eta$ der beiden Myonen. Verwenden Sie `Mu_eta`.

- Beide Myonen sollten unterschiedliche Ladung haben. Verwenden Sie `Mu_charge`.
- Fügen Sie alle Histogramme in ein zusätzliche `TPad` im `TCanvas` ein.

Folgendes Bild illustriert alle bisher verwendeten Variablen:



8 Fitten mit ROOT

ROOT beinhaltet das klassische Fit- oder Minimierungspaket **Minuit** (übernommen aus FORTRAN). Dies ist ein sehr mächtiges Programmpaket zur numerischen Minimierung; es ist Standard in Kern- und Teilchenphysik, wird aber auch ausserhalb der Physik verwendet (CERN Software für friedliche Zwecke frei verfügbar, Minuit ist das am meisten verwendete CERN Programm ausserhalb der Physik, abgesehen vom WWW ...).

Mittlerweile weitere Varianten für Minimierung in ROOT: *Minuit2*, *Fumili*

Sowohl für Histogramme als auch für (x, y) Wertepaare mit Fehlern ist das Fitten eine integrierte Methode der jeweiligen Klasse (*TH*, *TGraph*).

Besonders komfortabel geht es mit dem eingebauten FitPanel, mit dem man verschiedene Funktionen auswählen kann, sowie Fit-Optionen einstellen und den Fit-Range begrenzen kann.

Current selection: ha:TH1D

General | Minimization

Function

Predefined: Operation: Nop Add Conv

Selected: gaus

Fit Settings

Method:

Linear fit Robust: No Chi-square

Fit Options

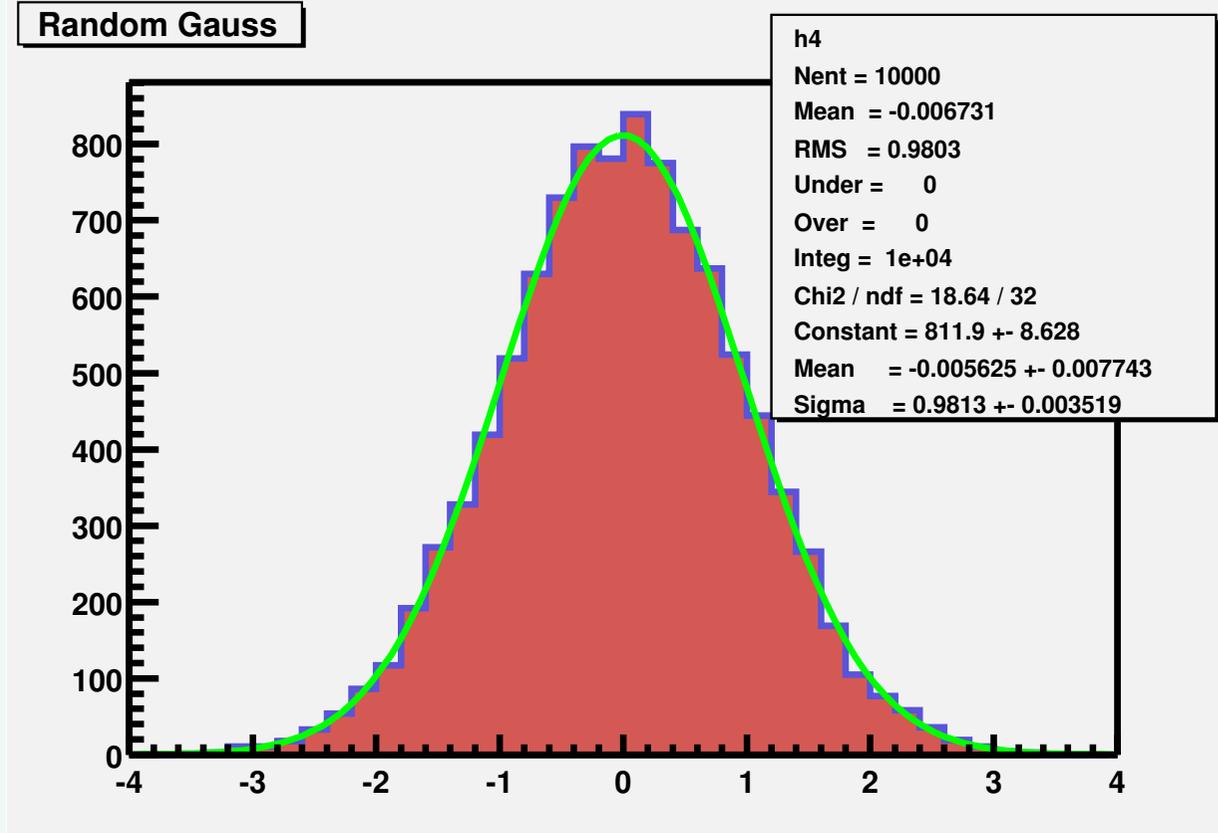
Integral Use range
 Best errors Improve fit results
 All weights = 1 Add to list
 Empty bins, weights=1

Draw Options

SAME No drawing Do not store/draw

X:

LIB Minuit MIGRAD Itr: 5000 Prm: DEF



8.1 Fitten von (x, y) Wertepaaren mit TGraphErrors

Mit der ROOT Klasse TGraphErrors kann man Wertepaare mit Fehlern darstellen und Fitten:

```
double xv[5] = { 1., 2., 3., 4., 5. };           1
double yv[5] = { 2.8, 3.7, 5.4, 6.1, 8.0 };     2
double ey[5] = { 0.5, 0.5, 0.5, 0.5, 0.5 };    3
double ex[5] = { 0.01, 0.01, 0.01, 0.01, 0.01 }; 4
TGraphErrors *tg = new TGraphErrors( 5, xv, yv, ex, ey ); 5
tg->Draw("AP");                                     6
```

Fitten anschliessend interaktiv

- Fitpanel: *mit rechter Maustaste auf Graph klicken, FitPanel auswählen*
- Kommandozeile/Programm:
`tg->Fit("pol1");`

Fitparameter, Fehler und Kovarianz kann man nach dem Fit über die ROOT Fitter Klasse `TVirtualFitter` abrufen:

```
TVirtualFitter *ft = TVirtualFitter::GetFitter();           1
// get fit parameter + error                               2
cout << ft->GetParameter(0) << " +- " << ft->GetParError(0) << endl; 3
cout << ft->GetParameter(1) << " +- " << ft->GetParError(1) << endl; 4
// get covariance                                         5
cout << ft->GetCovarianceMatrixElement(0,1) << endl;           6
```

8.2 Fitten von Histogrammen

Mit den ROOT Histogramm Klassen (TH1F, ...) kann man 1-dim Häufigkeitsverteilungen darstellen und fitten:

```
// create histo                                     1
int nbins = 50;                                       2
TH1F * hn = new TH1F("h4","Random Gauss",nbins,-4,4); 3
for ( int i = 0; i<10000; i++ ) {                   4
    hn->Fill(gRandom->Gaus());                          5
}                                                       6
hn->Draw();                                             7
// direct fit                                       8
hn->Fit("gaus");                                       9
// retrieve fit-function                             10
TF1 *fit = hn->GetFunction("gaus");                   11
// retrieve sigma                                    12
double sigma = fit->GetParameter(2);                 13
```

Interaktives Fitten analog

- Fitpanel: *mit rechter Maustaste auf Graph klicken, FitPanel auswählen*
- Kommandozeile/Programm ...

Standard ist ein χ^2 Fit :

Für Histogramme wird dabei als Fehler für ein Bin mit N Einträgen ein Gausscher Fehler \sqrt{N} angesetzt.

- Diese Näherung ist ausreichend für genügend grosses $N \geq 10$, d.h. man braucht genügend Daten dass (fast) alle Kanäle des Histogramms dies erfüllen
- Ansonsten mehrere Probleme:
 - grundsätzlich: Gauss-Näherung nicht mehr adäquat (Einträge Poisson-verteilt)
 - praktisch: Kanäle mit Null Einträgen werden beim Gauss-Fit ignoriert (keine sinnvolle Gauss Fehlerannahme möglich)
 - ⇒ Erzeugt Bias für Fit-Parameter, Fluktuationen nach unten (0) werden tendenziell ignoriert.

8.3 Maximum Likelihood Fit für Häufigkeitsverteilungen

Die optimale Methode zum Fitten einer Häufigkeitsverteilung an die Daten x_i ist ein **Unbinned Maximum Likelihood Fit** $\ln \mathcal{L} = \sum_{i=1}^n \ln p(x_i, \vec{a})$ bei dem jeder Datenwert explizit verwendet wird.

• Die Information wird voll ausgenutzt, keine Verluste durch binning

• Funktioniert auch bei geringer Statistik (wenig Messwerte)

Allerdings gibt es auch ein paar Nachteile:

- Erfordert direkte Programmierung entsprechender Minuit Funktion \Rightarrow Nächstes Mal.
- Konzeptionell: Keine direkte Kontrolle über Güte des Fits, man kann im Prinzip jede beliebige Funktion $p(x, \vec{a})$ fitten, egal ob sie die Verteilung der Daten beschreibt oder nicht. (Das resultierende \mathcal{L} am Minimum ist kein Mass für die Qualität des Fits, im Gegensatz zum χ^2)
- Bei großer Zahl von Messwerten sehr rechenintensiv ($\propto N$)
- Keine direkte Visualisierung möglich

Die letzten beiden Punkte kann man umgehen durch einen **Binned Maximum Likelihood Fit**:

- Ähnlich wie beim χ^2 Fit fasst man Messwerte in Bins zusammen. (Rechenzeit $\propto N_{bins}$)
- Die Zahl der Einträge in den einzelnen Bins ist Poisson-verteilt.

$$\ln \mathcal{L}(n_1, n_2, \dots, n_n | \vec{a}) = \sum_{bins} \left(\ln \frac{\mu_i^n e^{-\mu}}{n_i!} \right)$$

wobei μ sich aus der anzupassenden Verteilung berechnet:

$$\mu = N \int_{\Delta x_i} p(x | \vec{a})$$

- Im Vergleich zum χ^2 fit funktioniert das auch noch gut bei geringer Statistik, d.h. < 5 Einträgen pro Bin. Auch leere Bins mit $n=0$ werden im fit korrekt berücksichtigt, was bei χ^2 fits nicht sinnvoll möglich ist.

⇒ Übungen

8.4 Definition der Fit-Funktion

Einige Funktionen sind vordefiniert in ROOT:

- Polynome bis zum 9. Grad
- Gauss
- Exponentialfunktion
- Landau Verteilung
- Über FitPanel können Funktionen kombiniert werden:
z.B. `Gaus + pol2`

Fitten mit Benutzer definierten Funktionen

Für viele Fälle kann man auf weitere in ROOT eingebaute Funktionen zurückgreifen.

- `TF1 *myfit = new TF1("myfit", "[0]*sin(x) + [1]*exp(-[2]*x)", 0, 2);`
- Auswahl `user` im FitPanel und Eintrag `myfit` in Textfeld
- Oder Kommandozeile `hist->Fit("myfit")`
- Meist müssen brauchbare Startwerte für den Fit vorgegeben werden:
`myfit->SetParameters(1, 0.05, 0.2);`

Und für den allgemeinen Fall kann man eigene C++-Funktionen definieren:

```
double BreitWig( double *x, double *par)           1
{                                                  2
    // Breit- Wigner function                    3
    double mw = par[0], gw = par[1], spk=par[2], mw2, gw2, eb2; 4
                                                    5

    mw2 = mw*mw;                                  6
    gw2 = gw*gw;                                  7
    eb2 = x[0]*x[0];                              8
    return( spk * gw2*mw2 / ( pow( eb2 - mw2, 2 ) + mw2 * gw2 ) ); 9
    // return( spk * gw2*mw2 / ( pow( eb2 - mw2, 2 ) + eb2*eb2/mw2 * gw2 ) ); 10
}                                                  11
void bwfit()                                       12
{                                                  13
    const int np = 7;                              14
    double xv[np] = { 88.396, 89.376, 90.234, 91.238, 92.068, 93.080, 93.912 }; 15
    double *ex = 0;                                 16
    double yv[np] = { 6.943, 13.183, 25.724, 40.724, 27.031, 12.273, 6.980 }; 17
    double ey[np] = { 0.087, 0.119, 0.178, 0.087, 0.159, 0.095, 0.064}; 18
```

```
TGraphErrors *tg = new TGraphErrors( 7, xv, yv, ex, ey );           19
tg->Draw("AP");                                                  20
TF1 *bw = new TF1("bw",BreitWig, 70, 110, 3);                    21
// set parameters                                               22
bw->SetParameters(80.,2. , 10.);                                  23
// do the fit                                                    24
tg->Fit("bw");                                                    25
// retrieve fit function                                         26
TF1 *f = bw;                                                     27
}                                                                    28
```

8.5 Multi-Parameter Fit und Korrelationen

Wenn mehrere Parameter simultan gefittet werden ist es wichtig neben den Werten und Fehlern der Parameter auch die Korrelationen zu untersuchen.

In ROOT sind diese Informationen prinzipiell verfügbar, allerdings etwas versteckt und nicht direkt über das FitPanel abrufbar.

Für Fits die mit Minuit gemacht werden liefert folgende Funktion die Kovarianz- bzw. Korrelationsmatrix:

```

TMatrixD * GetCovMat( int ndim, bool print = false, bool corrmatrix = false )      1
{                                                                                   2
    TMatrixD td(ndim,ndim);                                                         3
    gMinuit->mnemat(td.GetMatrixArray(),ndim);                                       4
                                                                                       5
    if ( corrmatrix ) {                                                             6
        for ( int i=0; i<ndim; i++ ) {                                             7
            for ( int j=i+1; j<ndim; j++ ) {                                       8
                double cor = td(i,j) / sqrt( td(i,i) * td(j,j) );                 9
                td(i,j) = cor;                                                     10
                if ( i != j ) { // copy off diag elem                             11

```

```
        td(j,i) = td(i,j);           12
    }                                13
    }                                14
    td(i,i) = 1.0;                   15
    }                                16
}                                     17
if ( print ) td.Print();           18
                                     19

return ( new TMatrixD(td));       20
}                                     21
                                     22
```

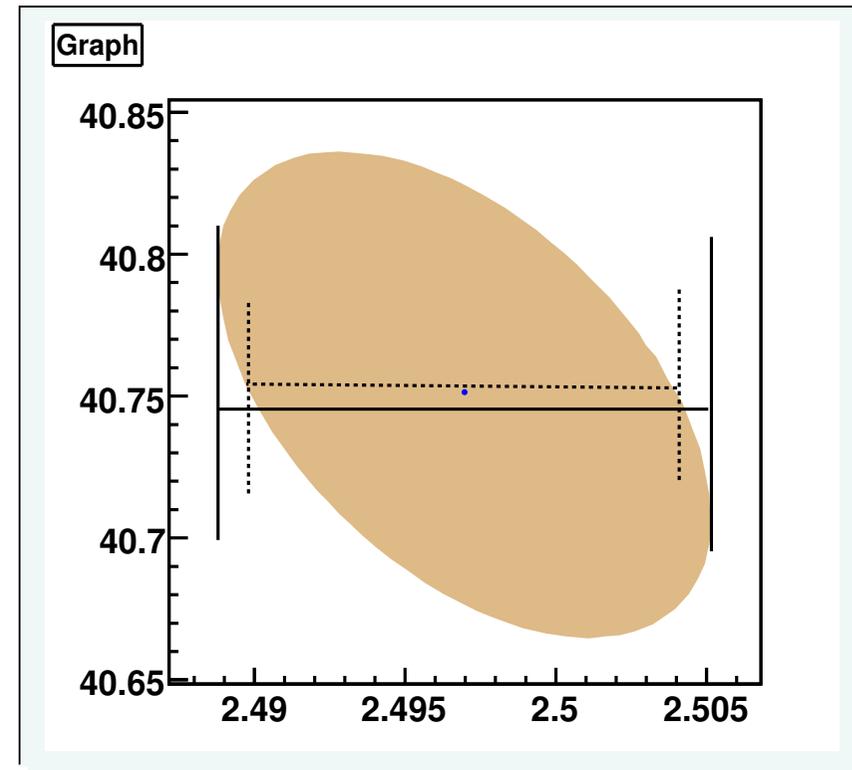
C/C++ Tricks:

- `TMatrixD` ist Matrix Klasse mit double als Elemente, viele nützliche Funktionen zum Bearbeiten, Drucken, Invertieren, ..., von Matrizen.
`TMatrixD td(ndim, ndim)` definiert $ndim \times ndim$ Matrix.
- `gMinuit` ist **globaler Pointer** auf aktuelles `TMinuit` Objekt.
- `gMinuit->mnemat(...)` ruft Minuit Funktion, die die Kovarianzmatrix der gefitteten Parameter im übergebenen Array (`td.GetMatrixArray()`) speichert.

Contour-Plot

Eine gute Visualisierung der Korrelation erlauben die sog. Contour-Plots, die die 1-sigma (oder n-sigma) Ellipse für zwei Fit-Parameter zeigen.

- Der Fehler eines Parameters wird durch Korrelation erhöht, die volle Breite bzw Höhe der Contour-Ellipse entspricht dem Fehler
- An der Ellipse lässt sich aber auch leicht ablesen wie stark sich der Fehler verringert wenn man einen Parameter fixiert (gestrichelte Linie)



In Minuit Funktion `gMinuit->Contour(...)` , die direkt als TGraph-Objekt und zum Zeichnen der Contour benutzt werden kann:

```
void plotContour( int i, int j )           1
{                                           2
    // make contour plot for two vars:     3
    TGraph *gr2 = (TGraph*)gMinuit->Contour(80,i,j); 4
    gr2->SetFillColor(42);                 5
    gr2->Draw("alf");                      6
                                           7
    // central value                       8
    double pi, dpi;                        9
    double pj, dpj;                       10
    gMinuit->GetParameter( i, pi, dpi ) ;  11
    gMinuit->GetParameter( j, pj, dpj ) ;  12
    TMarker *marker = new TMarker(pi,pj,20); 13
    marker->Draw();                        14
}                                           15
```

8.6 Fitten mit MINUIT

Das direkte Fitten mit ROOT ist beschränkt auf

- unkorrelierte Messungen
- χ^2 fit
- binned Maximum-Likelihood fit

Wenn das nicht reicht muss man innerhalb von ROOT MINUIT direkt aufrufen:

```
// initialize TMinuit 1
TMinuit *gMinuit = new TMinuit(npar); 2
// Tell Minuit the function 3
gMinuit->SetFCN(fcn); 4
// Start Werte fuer Minuit 5
gMinuit->mnparm(0, "a", 1.3, 0.1, 0.1, 5., ierflg); 6
// minimization 7
gMinuit->mnexcm("MIGRAD", arglist ,2,ierflg); 8
```

In der Funktion `fcn` muss man dann selbst χ^2 oder \mathcal{L} berechnen, wobei `fcn` folgende Argumente bekommt:

```
void fcn(Int_t &npar, Double_t *gin, Double_t &f,  
Double_t *par, Int_t iflag)
```

Wichtig sind

- pointer/array `par`, enthält die aktuellen Werte der Parameter.
- `f`, damit wird das berechnete χ^2 oder \mathcal{L} zurückgegeben.

Die Syntax zur Minuit Verwendung ist ein bisschen speziell, weder C noch C++ sondern an FORTRAN angelehnt. Minuit in FORTRAN geschrieben, ROOT Implementation ist direkte f2c Übersetzung.

8.7 Unbinned Maximum-Likelihood Fit

- log-likelihood für Daten und pdf in Abhängigkeit der FitParameter berechnen
- Minuit **minimiert**, deshalb Vorzeichen von *Max-LogLikelihood* rumdrehen
- Default Fehler Definition für χ^2 Fit \Rightarrow Faktor 0.5 für log-likelihood

```

const int N = 40;
double arr[N];
void AsyRnd( bool outp = true )
{
    // get random number according to asymmetry function
    TF1 *f1 = new TF1("f1","3/8*(1+0.5*x+x*x)", -1, 1 );

    for ( int i =0; i<N; i++ ) {
        // arr[i] = f1->GetRandom();
        arr[i] = gRandom->Rndm()-0.1;
        if ( outp ) cout << " i = " << i << " x = " << arr[i] << endl;
    }
}

```

```
double fEval( double x, double p[]) 14
{ // calculate asy pdf for given x and asy par 15
  double val = 3./8*(1.+p[0]*x+x*x); 16
  return val; 17
} 18
void fcn(Int_t &npar, double *gin, double &f, double *par, int iflag) 19
{ // calculate log likelihood for generated values arr[i] and asy par 20
  double logl = 0.; 21
  for ( int i =0; i<N; i++ ) { 22
    double fval = fEval( arr[i], par ) ; 23
    if ( fval < 1e-10 ) fval = 1e-10; // protect against 0/neg 24
    logl += log( fval ); 25
  } 26
  // return negative log-lik divided by 2 for error def 27
  f = -logl/2.; 28
} 29
void fAsyMn( ) 30
{ 31
  // create asy numbers 32
  33
```

```
AsyRnd(); 34
// initialize TMinuit 35
const int npar = 1; 36
gMinuit = new TMinuit(npar); 37
// Tell Minuit the function 38
gMinuit->SetFCN(fcn); 39
// Start Werte fuer Minuit 40
int ierflg; 41
gMinuit->mnparm(0, "Asy", 0.1, 0.01, 0., 0., ierflg); 42
// minimization 43
double arglist[3] = { 1000., 1., 0. }; 44
gMinuit->mnexcm("MIGRAD", arglist ,1,ierflg); 45
46
double par, epar; 47
gMinuit->GetParameter( 0, par, epar ) ; 48
cout << "Asy " << par << " +- " << epar << endl; 49
} 50
```

8.8 Fitten korrelierter Daten

$$\chi^2 \equiv \sum_{i=1}^N \frac{(y_i - f(x_i, \vec{a}))^2}{(\sigma_i)^2}$$

Die einfache Formel

gilt nur für unkorrelierte Messwerte, es gibt aber häufig Korrelationen zwischen Messwerten, z.B.

- gemeinsame Normierungsunsicherheit
- gemeinsamer Nullpunkt/Startwert
- andere systematische Effekte

Dann muss die 'Kovarianzmatrix' für den Fit benutzt werden. Für n Werte y_1, y_2, \dots, y_n beschreibt die $n \times n$ Kovarianzmatrix \mathcal{C} Fehler und Korrelationen der einzelnen Werte:

$$\mathcal{C} = \begin{pmatrix} \sigma_{y_1}^2 & \text{COV}_{y_1, y_2} & \dots & \text{COV}_{y_1, y_n} \\ \text{COV}_{y_1, y_2} & \sigma_{y_2}^2 & \dots & \text{COV}_{y_2, y_n} \\ \dots & \dots & \dots & \dots \\ \text{COV}_{y_1, y_n} & \text{COV}_{y_2, y_n} & \dots & \sigma_{y_n}^2 \end{pmatrix}$$

mit $\text{COV}_{y_i, y_j} = \rho_{ij} \sigma_{y_i} \sigma_{y_j}$.

Das χ^2 wird dann gemäss der allgemeinen Formel

$$\chi^2 \equiv \vec{\Delta}^T \cdot \mathcal{C}^{-1} \cdot \vec{\Delta}$$

berechnet werden

(\mathcal{C}^{-1} = invertierte Kovarianzmatrix)

($\vec{\Delta}$ = Vektor der Residuen : $(y_i - f(x_i, \vec{a}))$)

Zum Erstellen und Invertieren der Kovarianzmatrix \mathcal{C} empfiehlt sich die ROOT Klasse [TMatrixD](#).

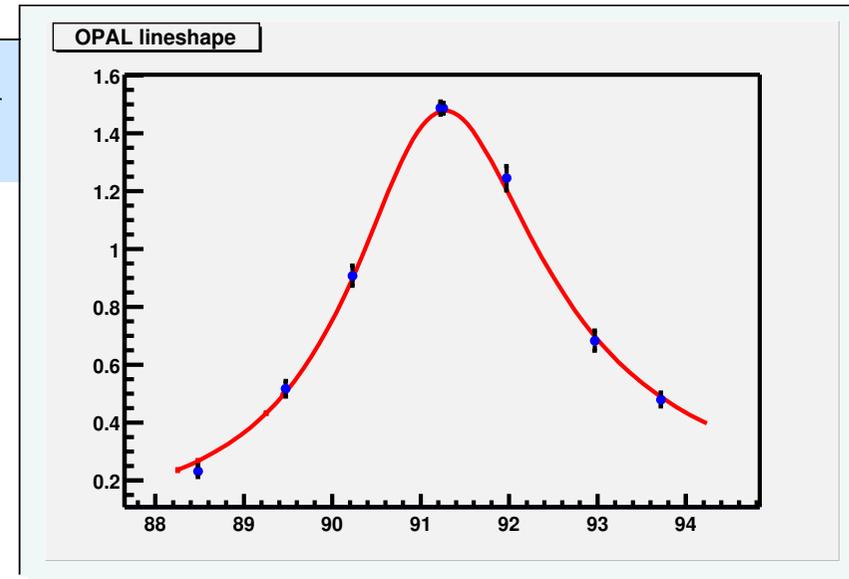
Z^0 Resonanzkurve des Wirkungsquerschnitts

Gutes Beispiel für korrelierte Fehler ist Messung des Wirkungsquerschnitts bei LEP im Bereich der Z^0 Resonanz.

$$\sigma_f(E_{CM}) = \frac{N_{sel} - N_{BG}}{\varepsilon \int \mathcal{L} dt}$$

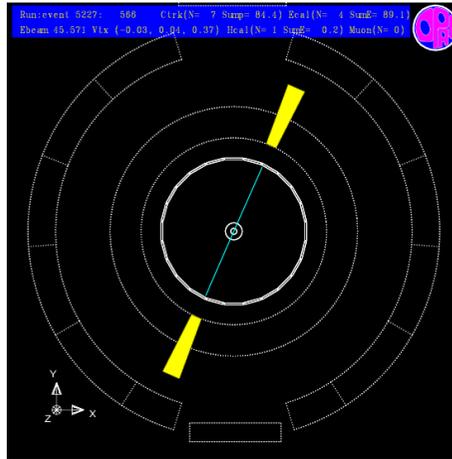
Experimentell:

- Statistische (=unabhängige) Fehler aus N_{sel}
- Systematische Fehler in y aus Selektion, Untergrund, Luminositätsmessung
- Systematische Fehler in x aus Unsicherheit in E_{CM}

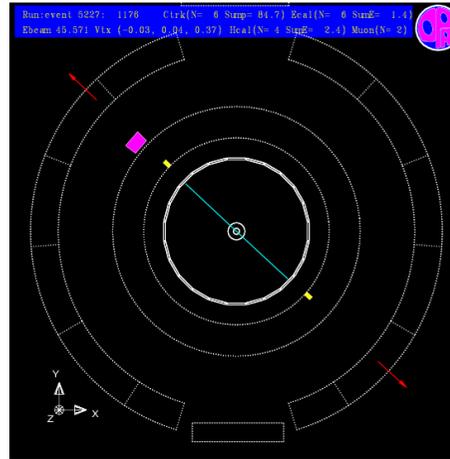


Klassische Messung, auch im Fortgeschrittenen Praktikum an der LMU !

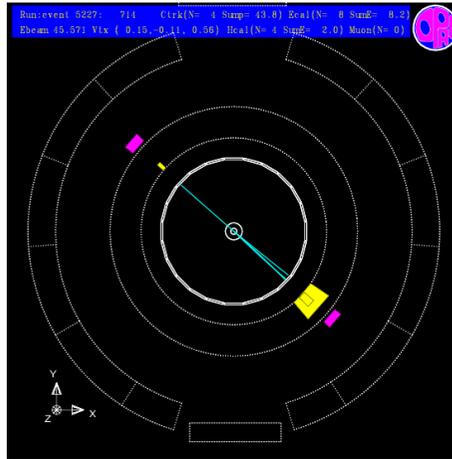
e^+e^-
(3.3%)



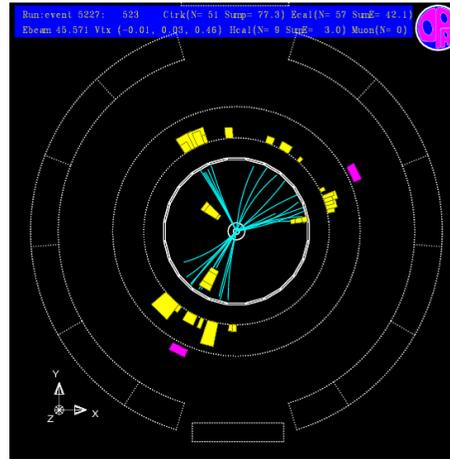
$\mu^+\mu^-$
(3.3%)



$\tau^+\tau^-$
(3.3%)



$q\bar{q}$
(70%)



Entsprechender Fit mit Minuit:

```
#include "TMatrix.h" 1
TMatrixD *cov; 2
double *val; 3
double *xval; 4
const int nv = 7; 5
void setInput() 6
{ // define input 7
  val = new double[nv]; 8
  xval = new double[nv]; 9
  cov = new TMatrixD(nv,nv); 10
  double xv[nv] = { 88.396, 89.376, 90.234, 91.238, 92.068, 93.080, 93.912 }; 11
  double yv[nv] = { 6.943, 13.183, 25.724, 40.724, 27.031, 12.273, 6.980 }; 12
  double ey[nv] = { 0.087, 0.119, 0.178, 0.087, 0.159, 0.095, 0.064}; 13
  double rsys = 0.004; // correlated relative syst uncertainty on yv 14
  for ( int i=0; i<nv; i++ ) { 15
    val[i] = yv[i]; 16
    xval[i] = xv[i]; 17
    cov(i,i) = ey[i]*ey[i]; // stat errors 18
```

```
for ( int j=i; j<nv; j++ ) { 19
    cov(i,j) = cov(i,j) + yv[i] * yv[j] * rsys * rsys; 20
    cov(j,i) = cov(i,j); 21
} 22
} 23

cov->Print(); 24
cov->Invert(); 25
cov->Print(); 26
} 27
} 28
double BreitWig( double *x, double *par) 29
{ 30
    // Breit- Wigner function 31
    double mw = par[0], gw = par[1], spk=par[2], mw2, gw2, eb2; 32
    33
    mw2 = mw*mw; 34
    gw2 = gw*gw; 35
    eb2 = x[0]*x[0]; 36
    return( spk * gw2*mw2 / ( pow( eb2 - mw2, 2 ) + mw2 * gw2 ) ); 37
    // return( spk * gw2*mw2 / ( pow( eb2 - mw2, 2 ) + eb2*eb2/mw2 * gw2 ) ); 38
}
```

```
} 39
void fcn(Int_t &npar, double *gin, double &f, double *par, int iflag) 40
{ 41
    // calculate Chi2 42
    double chi2 = 0.; 43
    for ( int i=0; i<nv; i++ ) { 44
        double fvali = BreitWig( &xval[i], par ); 45
        for ( int j=0; j<nv; j++ ) { 46
            double fvalj = BreitWig( &xval[j], par ); 47
            chi2 += ( fvali - val[i] ) * cov(i,j) * ( fvalj - val[j] ) ; 48
        } 49
    } 50
    // cout << "Chi2 = " << chi2 << endl; 51
    f = chi2; 52
} 53
void bwfitmnc() 54
{ 55
    setInput(); 56
    // initialize TMinuit 57
    const int npar = 3; 58
```

```
gMinit = new TMinuit(npar); 59
// Tell Minuit the function 60
gMinit->SetFCN(fcn); 61
// Start Werte fuer Minuit 62
int ierflg; 63
gMinit->mnparm(0, "mass", 85, 0.1, 0., 0., ierflg); 64
gMinit->mnparm(1, "width", 2., 0.1, 0., 0., ierflg); 65
gMinit->mnparm(2, "speak", 10., 0.1, 0., 0., ierflg); 66
// minimization 67
double arglist[3] = { 1000., 1., 0. }; 68
gMinit->mnexcm("MIGRAD", arglist ,1,ierflg); 69
70
// fcn( nparfit, &zero, &chi2, par, 5, 0 ); 71
// TGraphErrors *tg = new TGraphErrors( 7, xv, yv, ex, ey ); 72
//tg->Draw("AP"); 73
// central value 74
double par[npar], epar[npar]; 75
for ( int i=0; i<npar; i++ ) { 76
    gMinit->GetParameter( i, par[i], epar[i] ) ; 77
} 78
```

```
double ey[nv] = { 0.087, 0.119, 0.178, 0.087, 0.159, 0.095, 0.064};           79
for ( int i=0; i<nv; i++ ) {                                               80
    // double delta = (yv[i] - f->Eval(xv[i]));                               81
    double delta = (val[i] - BreitWig(&xval[i], par));                       82
    cout << val[i] << " +- " << ey[i] << " delta: " << delta << " " << delta/ey[i] << endl; 83
}                                                                             84
}                                                                             85
```

Berücksichtigung von Fehlern in der x-Koordinate

Bei Messungen von (x, y) Wertepaaren gibt es oft Fehler nicht nur in Δy sondern auch in Δx .

- Der übliche χ^2 Ansatz erlaubt nur Unsicherheiten/Kovarianz in y
- Trick: Über Steigung der Fit-Funktion kann man Fehler Δx in Fehler Δy transformieren:

$$\sigma_i^{y(x)} = \left. \frac{df(x, \vec{a})}{dx} \right|_{x=x_i} \sigma_i^x$$

- Iterativ vorgehen:
 - Erst Fit ohne Berücksichtigung der $\Delta x \Rightarrow$ Näherung für Fit-Funktion und Parameter
 - Dann Steigung der Fit-Funktion an den Stellen x_i bestimmen und Fehler zur Kovarianzmatrix addieren.
 - Meist genügt einer Iteration ...

```
#include "TMatrix.h" 1
TMatrixD *cov; 2
double *val; 3
double *xval; 4
const int nv = 7; 5
void setInput( bool deltax = true, double * par = 0 ) 6
{ // define input 7
  val = new double[nv]; 8
  xval = new double[nv]; 9
  cov = new TMatrixD(nv,nv); 10
  double xv[nv] = { 88.396, 89.376, 90.234, 91.238, 92.068, 93.080, 93.912 }; 11
  double yv[nv] = { 6.943, 13.183, 25.724, 40.724, 27.031, 12.273, 6.980 }; 12
  double ey[nv] = { 0.087, 0.119, 0.178, 0.087, 0.159, 0.095, 0.064 }; 13
  double rsys = 0.004; // correlated relative syst uncertainty on yv 14
  double esys = 0.007; // 7 GeV syst error on x=E (absolute) 15
  TF1 * f1; 16
  if ( deltax ) { 17
    // use as Root function 18
    TF1 * f1 = new TF1( "bw", BreitWig, 70, 100, 3 ); 19
    f1->SetParameters( par ); 20
```

```

}
for ( int i=0; i<nv; i++ ) {
    val[i] = yv[i];
    xval[i] = xv[i];
    (*cov)(i,i) = ey[i]*ey[i]; // stat errors
    for ( int j=i; j<nv; j++ ) {
        (*cov)(i,j) = (*cov)(i,j) + yv[i] * yv[j] * rsys * rsys;
        if ( deltax ) { // treat errors in x, use TF1 Derivative method
            double dydxi = f1->Derivative( xv[i] ); // slope at x[i]
            double dydxj = f1->Derivative( xv[j] ); // slope at x[j]
            (*cov)(i,j) = (*cov)(i,j) + dydxi * esys * dydxj * esys;
        }
        (*cov)(j,i) = (*cov)(i,j);
    }
}
cov->Print();
cov->Invert();
cov->Print();
}
...

```

```
void bwfitmnce() 41
{ 42
  setInput(false); 43
  // initialize TMinuit 44
  const int npar = 3; 45
  gMinuit = new TMinuit(npar); 46
  // Tell Minuit the function 47
  gMinuit->SetFCN(fcn); 48
  // Start Werte fuer Minuit 49
  int ierflg; 50
  gMinuit->mnparm(0, "mass", 85, 0.1, 0., 0., ierflg); 51
  gMinuit->mnparm(1, "width", 2., 0.1, 0., 0., ierflg); 52
  gMinuit->mnparm(2, "speak", 10., 0.1, 0., 0., ierflg); 53
  // minimization 54
  double arglist[3] = { 1000., 1., 0. }; 55
  gMinuit->mnexcm("MIGRAD", arglist ,1,ierflg); 56
  57
  double par[npar], epar[npar]; 58
  for ( int i=0; i<npar; i++ ) { 59
    gMinuit->GetParameter( i, par[i], epar[i] ) ; 60
```

```
} 61
// include x errors and iterate fit 62
setInput(true, par); 63
gMinuit->mnexcm("MIGRAD", arglist ,1,ierflg); 64
// accurate error matrix 65
gMinuit->mnexcm("HESSE", arglist ,1,ierflg); 66
... 67
} 68
```

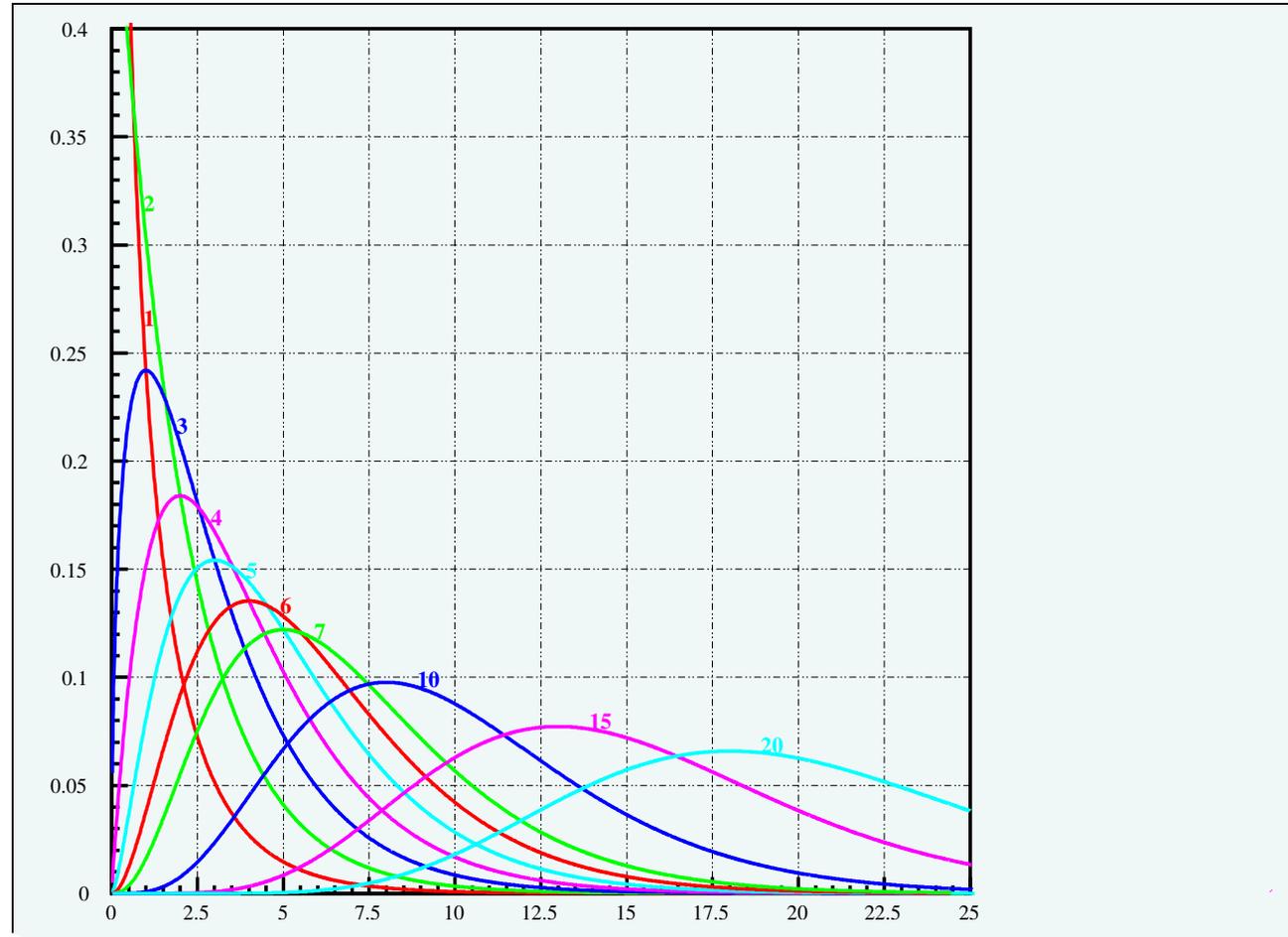
8.9 Güte des Fit – χ^2

Der Wert des χ^2 Fits am Minimum liefert auch gleichzeitig ein Mass für die Güte des Fits. Wenn die Unterschiede zwischen den gemessenen (y_i) und gefitteten ($f(x_i, \vec{a})$) Werten konsistent mit statistischen Schwankungen sind erwartet man ein χ^2 gemäss der χ^2 Funktion:

$$f(z, ndf) = \frac{z^{ndf-2} e^{-z^2/2}}{2^{-ndf/2} \Gamma(ndf/2)}$$

wobei ndf die Zahl der Freiheitsgrade ist, $ndf = N_{points} - N_{par}$.

Bei vielen Wiederholungen des Experiments bzw. Fits sollte das resultierende χ^2 einer solchen Verteilung folgen.



$$\chi^2 / ndf \approx 1$$

Als Faustregel kann man sich merken:

Deutlich grössere Werte sind ein klarer Hinweis auf konzeptionelle Probleme, entweder sind die Fehler unterschätzt oder die angepasste Funktion ist nicht geeignet zur Beschreibung der Daten. Kleine Werte deuten auf zu große Fehler hin, typisch sind falsch bestimmte statistische Fehler.

In ROOT:

- Funktion `TMath::Prob(double chi2, int ndf)`
- Liefert Wahrscheinlichkeit $\chi^2 \geq chi2$ bei ndf Freiheitsgraden zu bekommen, z.B.:

```
TMath::Prob( 1.0, 3) = 0.801
```

```
TMath::Prob( 3.0, 3) = 0.392
```

```
TMath::Prob( 8.5, 3) = 0.037
```

8.10 Güte des Fit – Maximum-Likelihood / Ensemble-Tests

(Log-)Likelihood Wert am Maximum liefert kein absolutes Maß für Güte des Fits.

- Wert hängt ab von Details der Verteilung und Parameter, kein **absoluter Gütefaktor** ableitbar.
- Wert ist aber spezifisch für konkrete Verteilung und Parametersatz \Rightarrow **relativer Gütefaktor** im Vergleich mit *ähnlichen* Datensätzen.

Ensemble-Tests

- Fitte Datensatz mit unbinned Maximum-Likelihood.
- Würfle mit gefitteten Parametern N Datensätze gleicher Grösse, wiederhole Fit und speichere resultierenden (Log-)Likelihood Wert.
- Bestimme **p-Wert**, das ist Anteil der gewürfelten Datensätze die einen kleineren (Log-)Likelihood Wert haben.
- \Rightarrow Wahrscheinlichkeit dass Datensatz mit Verteilung kompatibel ist, analog zu χ^2 p-Wert.

8.11 Summary – Fitten mit ROOT

- Fits in ROOT sehr einfach, interaktiv mit FitPanel, für (x,y) Paare und Histogramme bei unkorrelierten Messwerten.
- Große Flexibilität bei Definiton/Wahl der Fit-Funktion
- Komplexere Fit-Probleme, z.B. unbinned likelihood, korrelierte Fehler, erfordern direkte Verwendung von Minuit.
- Mehr Aufwand in Programmierung, aber damit volle Kontrolle über Fehlerbehandlung, Konstruktion der Kovarianzmatrix, Ablauf des Fits, Korrelationen der Parameter, u.v.m.

9 Übungsaufgaben

9.1 ROOT: Datenein-/ausgabe, Mittelwert

1. C++ I/O und Mittelwert

In der Datei `rohr1.dat` finden Sie eine Liste von Messungen der Drahtposition in verschiedenen Atlas Muon-Kammer Rohren. Lesen Sie die Zahlen ein:

In C++: `while (cin >>val) ;`

In Linux: `./myprog < rohr1.dat`

Berechnen Sie Mittelwert, kleinsten und grössten Wert.

2. Aufwärmübungen mit ROOT

- Initialisieren und Starten Sie ROOT
- Gehen Sie einige der Beispiele aus 1.2 durch

3. Rohrdaten mit ROOT

Lesen Sie die Daten aus `rohr1.dat` in ROOT ein.

```
ifstream data_file;  
data_file.open("rohr1.dat");  
while ( data_file >>val );
```

Erzeugen Sie ein Histogramm und Füllen die Werte ein. (Lösungsbeispiel: `readRohr1.C`)

Analog für `rohr2.dat` in ein 2-dim Histogramm ('scatter-plot').
`TH2F("h", "mytitle", nx, xlow, xhigh, ny, ylow, yhigh);`
`h->Fill(x, y)`

4. C++ Mittelwert und Standardabweichung

In der Datei `rohr1.dat` finden Sie eine Liste von Messungen der Drahtposition in verschiedenen Atlas Muon-Kammer Rohren. Lesen Sie die Zahlen ein:

In C++: `while ((cin >>val[i++]) != 0) ;`

In Linux: `./myprog < rohr1.dat`

Berechnen Sie Mittelwert, und Standardabweichung, sowie Median und 25/75% Quartile (Tip: Zahlen sortieren).

Beispiellösungen: `readRohr1.C` Root macro, liest Werte ein und füllt Histogramm.

`readRohr2.C` C++ Programm, liest Werte in vector ein, berechnet Mittelwert, Median, etc.

5. Korrelation

In `rohr2.dat` ist eine Liste mit jeweils zwei Messungen der Drahtposition. Berechnen Sie Mittelwerte, und Standardabweichungen, sowie den Korrelationskoeffizienten.

6. Zentraler Grenzwert Satz

Überprüfen Sie das Theorem, dass die Mittelwerte beliebiger Verteilungen normalverteilt sind für große n .

(a) Verwenden Sie gleichverteilte Zufallszahlen (`gRandom->Rndm()`) und testen Sie wie die

Verteilung von n abhängt.

```
void tclim( Int_t n = 12, Int_t ns = 1000 )           1
{                                                     2
  TH1F * hclim = new TH1F("hclim", "central limit test", 100, -5, 5); 3
  for ( Int_t i = 0; i<ns; i++ ) {                   4
    Float_t sum = 0;                                  5
    for ( Int_t j = 0; j<n ; j++ ) {                 6
      sum += gRandom->Rndm() - 0.5;                  7
    }                                                 8
    hclim->Fill( sum );                               9
  }                                                  10
}                                                    11
```

(b) Nehmen Sie statt der Gleichverteilung die Exponentialverteilung (`gRandom->Exp(1)`)

7. Verteilungen plotten

Erstellen Sie ROOT Macros zum Plotten von

- Binomial-verteilung in Abhängigkeit von p, n
- Poisson-verteilung in Abhängigkeit von μ

- Gauss-Verteilung in Abhängigkeit von μ, σ

(Tip: Die **TMath Klasse** stellt entsprechende Funktionen oder Hilfsmittel bereit: `TMATH::Gamma(n+1)`, `TMATH::Binomial(n,k)`, `TMATH::Poisson(x, mu)`, ...)

Vergleichen Sie die Binomial-Verteilung mit der entsprechenden Poisson-Verteilung für verschiedene n und analog Binomial vs Gauss und Poisson vs Gauss.

9.2 Zufallszahlen

1. **Erzeugen von Zufallszahlen** Ein gängiges Verfahren zur Erzeugung von Pseudo-Zufallszahlen ist die *lineare Kongruenzmethode*:

$$\mathcal{I}_{n+1} = (a \cdot \mathcal{I}_n + c) \bmod m$$

Grundsätzlich sind solche Folgen periodisch, die Periodenlänge ist im besten Falle m . Entscheidend für die Güte des Generators ist die Wahl der Parameter a und b , sowie des ‘Seed’-Werts \mathcal{I}_0 .

Im folgenden sollen die Eigenschaften von Generatoren mit verschiedenen Parametersets untersucht werden.

Dazu benötigt man zunächst einen Zufallsgenerator, den man einfach als C++ Klasse implementieren kann (file [MyRnd.C](#)).

Die Implementation des Zufallsgenerators als C++ Klasse hat wesentliche Vorteile gegenüber einer konventionellen Implementation als Funktion: alle Instanzen von **MyRnd** sind völlig unabhängig, d.h. man kann beliebig viele Zufallsgeneratoren mit unterschiedlichen Seeds und Parametern verwenden.

Einfache Tests von Zufallsgeneratoren sind die Überprüfung der Gleichverteilung in 1, 2 oder 3 Dimensionen.

Beispielprogramm **tRnd.C**:

Aufgabe a:

Testen Sie verschiedene Parameter für den Zufallsgenerator:

(a) $a = 137$, $c = 187$, $m = 256$

(b) $a = 193$, $c = 73$, $m = 256$

(c) $a = 65539$, $c = 0$, $m = 2147483648$

Als seed sollte man eine Zahl > 0 verwenden.

(a) und (b) sind 'kurze' Generatoren. Sie nutzen die Länge m voll aus, zeigen aber deutliche Unterschiede in ihrer Qualität in 2 D.

(c) ist der Generator **randu** von IBM, der in den 60er und 70er Jahren weitverbreitet war. Er hat zwar eine hohe Periode (2^{29}), aber $a = 2^{16} + 3$ ist lausig gewählt, schon in 3 D zeigen sich deutliche Abweichungen von der Gleichverteilung. Viele wissenschaftliche Arbeiten haben darunter gelitten

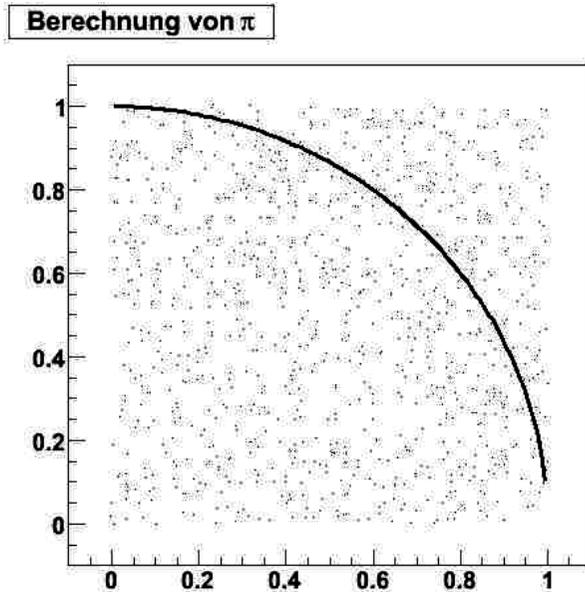
...

Zum Vergleich können Sie auch den ROOT Standard Generator mit `gRandom->Rndm()` betrachten.

2. Monte Carlo Berechnung der Kreiszahl π

Berechnen sie die Kreiszahl pi mit Hilfe der Monte Carlo Integration wie in der Vorlesung gesprochen.

- Verwenden sie hierzu die Tropfenmethode. Erzeugen sie zwei unabhängige gleichförmig verteilte Zufallszahlen mit:
- `TRandom ran`
- `ran.SetSeed("1")`
- `x = ran.Uniform()`
- Testen sie ob $r^2 = x^2 + y^2$ innerhalb oder außerhalb der Kreisfläche liegt
- Berechnen sie π für 100, 1000 und 10000 Tropfen und den zugehörigen Fehler.
- Füllen sie die erzeugten Zufallszahlen in ein Histogramm `histo = TH2F('test', 'Berechnung von #pi', 100, -0.1, 1.1, 100, -0.1, 1.1)`
- Lösungsbeispiel: `pi.C`, `pi.py`



3. Simulation des radioaktiven Zerfalls

Mit gleichverteilten Zufallszahlen lässt sich der radioaktive Zerfall leicht simulieren, die Wahrscheinlichkeit daß ein Kern zerfällt ist

$$p = \lambda \Delta t$$

In `tdecay.C` wird ein Zerfallsexperiment simuliert, wobei man die Zerfallskonstante λ , das Zeitintervall dt , die Gesamtzeit t und die Zahl der Nukleonen n_0 zu Beginn vorgeben kann.

Testen Sie zunächst das Programm für verschiedene Parameter. Dann ändern Sie das Programm so ab, daß Sie viele solcher Experimente durchführen und zusätzlich die Zahl der zerfallenen

Teilchen für einige Zeitintervalle histogrammieren.

```
// simulate radioactive decay 1
void tdecay( Float_t lambda = 0.5, Float_t dt = 0.01, Float_t t = 20, 2
    Int_t n0 = 1000 ) 3
{ 4
    TH1F * hdec = new TH1F("hdec", "radioactive decay", t/dt, 0, t); 5
    // loop over time 6
    for ( Float_t tnow = 0.; tnow < t; tnow += dt ) { 7
        // loop over remaining particles 8
        Int_t ndec = 0; 9
        for ( int ie = 0; ie < n0; ie++ ) { 10
            if ( gRandom->Rndm() < dt*lambda ) { 11
                ndec ++; 12
                hdec->Fill(tnow); 13
            } 14
        }
        n0 -= ndec; // subtract decayed particles 15
    } 16
} 17
```

4. Zusatzaufgaben:

Wie sieht die analytische Transformation aus, um Zufallszahlen gemäss $f(x) = 2 \cdot x$ bzw. e^{-x} zu erzeugen ? Probieren Sie es in ROOT.

Erzeugen Sie eine beliebige Zufallsverteilung nach dem Hit&Miss Verfahren, z.B. für die Breit-Wigner Funktion (**BreitWig.C**).

Hinweis: Funktionen als Macros

Neben der direkten Art Funktionen zu definieren, wie in

```
TF1 *f1 = new TF1("f1","sin(x)", 0, 10 );
```

kann man auch eine Funktion als Adresse angeben, z.B.:

```
// file BreitWig.C
Double_t BreitWig( Double_t *x, Double_t *par)
{
    // Breit- Wigner function
    Double_t mw = par[0], gw = par[1], mw2, gw2, eb2;
    mw2 = mw*mw;
    gw2 = gw*gw;
    eb2 = x[0]*x[0];
    return( gw2*mw2 / ( pow( eb2 - mw2, 2 ) + mw2 * gw2 ) );
}
```

```
root [0] .L BreitWig.C 11
root [3] TF1 *f1 = new TF1("f1",BreitWig,50,100,2) 12
root [4] f1->SetParameters( 80., 2.); 13
    root [5] f1->Draw(); 14
```

Als 5. Argument von `TF1(..)` muss die Zahl der Parameter stehen. Die Parameter können dann per `f1->SetParameters()` gesetzt werden.

Zufallszahlen die gemäss einer Funktion verteilt sind können in ROOT mit der Methode `TF1::GetRandom()` erzeugt werden, also z.B.:

```
root [5] f1->GetRandom()
```

9.3 Fehlerbestimmung

1. **Fehler I:** Welche Methode ist vorzuziehen: 10 Messungen mit einer Auflösung von 1 mm oder 1 Messung mit einer Auflösung von 0.2 mm ?
2. **Fehler II:** Eine elektrische Spannung U wird durch einen Strom $I = 1120 \pm 10\text{mA}$ und einen Widerstand $R = 1400 \pm 30\Omega$ bestimmt. Wie groß ist U und der zugehörige Fehler ?
3. **Fehler III:** Ein Strom I wird bestimmt durch eine Spannung $U = 45 \pm 1\text{V}$ und den Widerstand $R = 900 \pm 10\Omega$. Wie groß ist I und der zugehörige Fehler ?
4. **Fehlerfortpflanzung:** Sie messen 2 Grössen, die erste (x) hat Mittelwert 5 und $\sigma = 1$, die 2. (y) Mittelwert 10 und ebenfalls $\sigma = 1$. Die beiden Grössen haben eine Korrelation $\rho = 0.9$.
 - (a) Was ist σ der Differenz $\delta = x - y$ bzw. des Quotienten $r = x/y$?
 - (b) Folgendes Programm liefert entsprechende Zufallszahlen x und y . Probieren Sie's aus mit ROOT, und füllen Differenz bzw. Quotient in Histogramme.

```
void get2Rndm( double &x, double &y, double rho=0.9)           1
{                                                            2
```

```
double r1, r2;                                     3
gRandom->Rannor(r1,r2);                             4
x = r1 + 5.;                                        5
y = rho * r1 + sqrt(1.-rho*rho) * r2 + 10.;        6
}                                                    7
```

9.4 Mittelwerte

1. **Fehler des Mittelwerts** Das folgende Root-Makro enthält ein kleines Toy-MC um Fehler des Mittelwerts bei Gleichverteilung zu studieren.

- Untersuchen Sie die Streuung des Mittelwerts in Abhängigkeit der Sample Grösse n .
- Nehmen Sie als Schätzer $(x_{max} - x_{min})/2$ und vergleichen Sie die Streuung.

```

TH1* hmean( int n=50, int ns=100 )           1
{                                             2
  // fill generated values in histo         3
  TH1D * hm = new TH1D("hm", "Mittelwert von Rndm", 200, -0.5, 0.5); 4
  for ( int i =0; i<ns; i++ ) {             5
    double sum = 0;                          6
    for ( int j =0; j<n; j++ ) {           7
      sum += gRandom->Rndm() - 0.5; // gleichverteilung -0.5 .. 0.5 8
    }                                       9
    hm->Fill( sum / n );                    10
  }                                       11
  hm->Draw();                               12
}                                           13

```

-
2. **Getrimmter Mittelwert** Im C++ Makro `hTrimMean.C` finden Sie eine Vorlage zur Berechnung des Trimmed-Mean.

Untersuchen die Streuung des Mittelwerts als Funktion von r für verschiedene verteilungen (Gauss, Doppelt-Exponential, Cauchy, ...).

3. **Log-Likelihood für Asymmetrie** In `fAsy.C` ist der Code für das vorgestellt Beispiel zur Log-Likelihood Berechnung für die Asymmetrie.

Untersuchen Sie die log-Likelihood Kurve für verschiedenen Fälle, variieren Sie die Zahl der Ereignisse.

9.5 Fits (I)

1. Fits von Datenpunkten

Suchen Sie ein passendes Polynom um die beiden unten gegebenen Datensets vernünftig zu fitten, d.h. mit $\chi^2/ndf \approx 1$.

```
// 1. Datensatz                                     1
{                                                     2
Float_t xarr[16] = { 0.5, 1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5, 9.5,   3
                    10.5, 11.5, 12.5, 13.5, 14.5, 15.5};                4
Float_t yarr[16] = {0.731596, 3.11894, 0.368688, 5.24658, 4.34652,       5
                    7.10241, 9.99553, 9.5674, 11.0389, 13.5178, 11.9335,  6
                    14.9933, 15.1822, 18.3361, 17.5097, 21.8429};       7
Float_t exarr[16] = { 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1,      8
                    0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1 };              9
Float_t eyarr[16] = { 0.701041, 0.668213, 1.24997, 1.77685, 0.804879,   10
                    1.48788, 0.945508, 0.508232, 0.817365, 1.5927,     11
                    1.34098, 1.3508, 0.914614, 1.897, 1.66663, 1.42966}; 12
TGraphErrors *tg = new TGraphErrors( 16, xarr, yarr, exarr, eyarr );    13
tg->Draw("AP");}                                                    14
// 2. Datensatz                                                  15
```

```
{ 16
Float_t xarr[16] = { -3.5, -2.5, -1.5, -0.5, 0.5, 1.5, 17
                   2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5, 9.5, 10.5, 11.5}; 18
Float_t yarr[16] = {219., 155., 89., 11., 19
                   54., 80., 48., 5., -47., 20
                   -64., -101., -125., -124., 13., 21
                   239., 525. }; 22
Float_t exarr[16] = { 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 23
                    0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1 }; 24
Float_t eyarr[16] = { 65., 41, 30., 26., 26., 24., 25
                    18., 4., 17., 33., 45., 26
                    49., 38., 6., 74., 160.}; 27
TGraphErrors *tg = new TGraphErrors( 16, xarr, yarr, exarr, eyarr ); 28
    tg->Draw("AP");} 29
```

2. Eichkurve und Fehlerfortpflanzung

Sie kalibrieren einen Detektor mit Strahlen bekannter Energie von 60, 80, 100 GeV und erhalten jeweils

59.73 ± 0.35 , 80.64 ± 0.38 , $100.51 \pm 0.42 GeV$.

Anschliessend messen Sie für ein Teilchen eine Energie

$E_{mess} = 83 GeV$.

Was ist die wahre Energie und ihre Unsicherheit, $E_{true} \pm \Delta$?

```

void linefit()                                     1
{                                                  2
  float xv[3] = { 60., 80, 100. };               3
  float yv[3] = { 59.73, 80.64, 100.51 };        4
  float ey[3] = { 0.35, 0.38, 0.42 };           5
  float ex[3] = { 0.01, 0.01, 0.01 };           6
  TGraphErrors *tg = new TGraphErrors( 3, xv, yv, ex, ey ); 7
  tg->Draw("AP");                                 8
}                                                  9

```

Machen Sie einen Geradenfit: $E_{mess} = a + b \cdot E_{true}$.

Durch umkehren $E_{true} = -a + 1/b \cdot E_{mess}$ bestimmen Sie die wahre Energie, für Δ müssen Sie

die Fehlerfortpflanzung bemühen, d.h. Sie brauchen die Kovarianzmatrix für a und b .
(Fitoption "`v`" oder `verbose` im Fit-panel)

3. Fit in Subranges und mit selbstdefinierten Funktionen

Die Funktion `fbw.C` erzeugt ein Histogramm mit Zufallszahlen gemäss einer Breit-Wigner Verteilung.

```
void fbw() 1
{ 2
    Double_t ecm = 200, mw = 80; 3
    Double_t gamma_w = 2; // W Zerfallsbreite 4
    Double_t eb, beta, gamma; 5
    Double_t xrnd; 6
    // initialize 7
    eb = ecm/2.; // beam energy 8
    // define Breit-Wigner function 9
    TF1 *fbw = new TF1("fbw", BreitWig, 70, 90, 3); 10
    // set the parameters 11
    fbw->SetParameters( mw, gamma_w,1.); 12
    TH1F * hbw = new TH1F("hbw", "Breit Wigner", 100, 70, 90); 13
    for ( Int_t i = 0; i < 1000; i++ ) hbw->Fill(fbw->GetRandom()); 14
} 15
Double_t BreitWig( Double_t *x, Double_t *par) 16
```

```
{ 17
// Breit- Wigner function 18
Double_t mw = par[0], gw = par[1], mw2, gw2, eb2; 19
mw2 = mw*mw; 20
gw2 = gw*gw; 21
eb2 = x[0]*x[0]; 22
return( par[2]*gw2*mw2 / ( pow( eb2 - mw2, 2 ) + mw2 * gw2 ) ); 23
} 24
```

Versuchen Sie zunächst das Histogramm mit einer Gauss-Verteilung zu fitten. Variieren Sie dabei den Fit-range bis die Verteilung annähernd durch einen Gauss-Fit beschrieben wird.

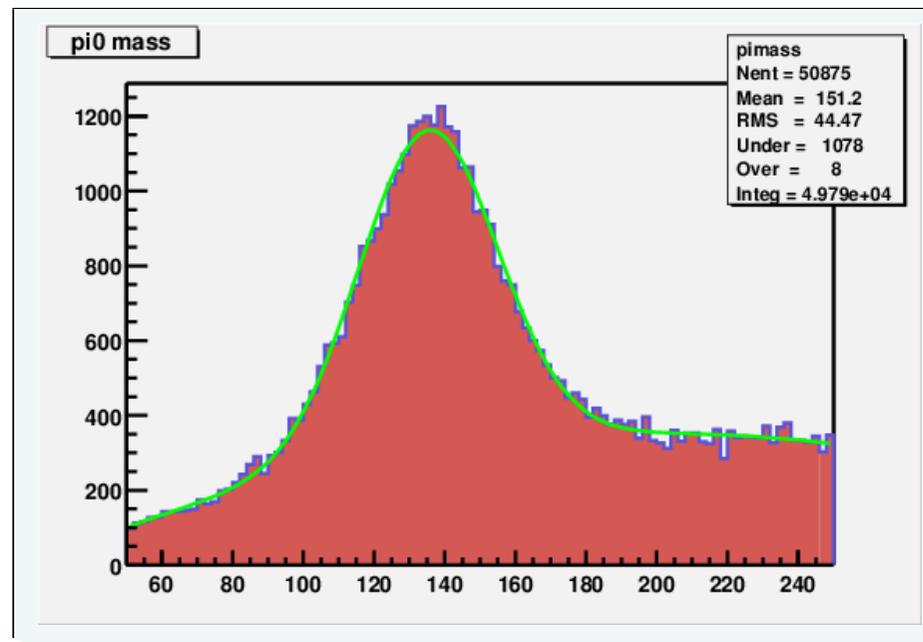
Anschliessend fitten Sie direkt die Breit-Wigner Funktion:

```
ROOT > fbw->SetParameters(83, 3, 100);
ROOT > hbw->Fit("fbw");
```

Testen Sie ob das Fit Resultat von den Startwerten abhängt.

4. Massenspektrum

In der Datei `pi0.dat` sind Daten von einem Experiment, in dem der Zerfall des neutralen Pions in zwei Photonen gemessen wurde $\pi^0 \rightarrow \gamma\gamma$. Der Datensatz enthält die invariante Masse für alle möglichen paarweisen Kombinationen der Photonen. Photonen entstehen auch in anderen Zerfällen, d.h. nur ein Teil der Werte ist ein echter π^0 Zerfall, der Rest ist Untergrund den man subtrahieren muss.



Bestimmen Sie die π^0 Masse und finden Sie heraus wieviele π^0 im Datensatz enthalten sind Dazu müssen Sie einen kombinierten Fit machen, der den Untergrund mit einem Polynom beschreibt und

den eigentlichen Signal peak mit einer Gauss Funktion.

```
TF1("pfit", "gaus(0) + pol2(3)", 50, 250)
```

In diesem Fall müssen Sie Startwerte für die Parameter vorgeben; fitten Sie zunächst in sub-ranges einen Gauss bzw. Polynom um vernünftige Anfangswerte zu erhalten.

5. χ^2 fit und Maximum Likelihood Fit

Die Funktion `gausf2.C` füllt ein Histogramm mit Zufallszahlen gemäss einer Gauss Verteilung.

Anschliessend wird ein χ^2 oder Maximum Likelihood Fit durchgeführt.

Das ganze wird mehrmal (*nit*) durchgeführt und die jeweils resultierende gefittete Breite in ein Histogramm gefüllt.

Variieren Sie die Zahl der Bins oder der Einträge und vergleichen Sie die Ergebnisse der beiden Fit Methoden, insbesondere wenn nur wenig Einträge pro bin gemacht werden.

```

gausf2( bool logl = false, int niter = 100, int nrnd = 100, int nbins = 40 )      1
{                                                                                   2
    // logl : flag for log-likelihood / chi2 fit                                    3
    // niter: number of iterations                                                 4
    // nrnd : number of random entries for histo                                  5
    // nbins: number of bins                                                       6
    double sig;                                                                    7
    // create histo                                                                8
    TH1D * h4 = new TH1D("h4","Random Gauss",nbins,-4,4);                          9
    // hist for sigma                                                            10
    TH1D * sigma = new TH1D("sigma","sigma from gaus fit",20,0.5,1.5);            11
    for ( i =0; i<niter; i++ ) {                                                  12

```

```
// reset histo contents 13
h4->Reset(); 14
// fill histo 15
for ( int j = 0; j<nrnd; j++ ) { 16
    h4->Fill(gRandom->Gaus()); 17
} 18
if ( logl ) { // likelihood fit 19
    h4->Fit("gaus","lq"); 20
} 21
else { // Chi2 fit 22
    h4->Fit("gaus","q"); 23
} 24
// h4->Draw(); 25
// get sigma from fit 26
TF1 *fit = h4->GetFunction("gaus"); 27
sig = fit->GetParameter(2); 28
sigma->Fill(sig); 29
} 30
sigma->Draw(); 31
} 32
```


9.6 Fits (II)

1. Mitteln von korrelierten Messungen

In `avMinu.C` wird mit Hilfe von Minuit der Mittelwert von 2 korrelierten Messungen bestimmt. Im Beispiel sind die Werte 8.0 und 8.8, mit korreliertem Fehler von 10 %.

Führen Sie die Mittelung aus

(a) mit unabhängigem Fehler von 20 % (`stat = 0.2`)

(b) mit unabhängigem Fehler von 2 % (`stat = 0.02`)

Diskutieren Sie die Ergebnisse.

2. Unbinned likelihood Fit

In `fAsyMn.C` wird ein Maximum Likelihood Fit für die Winkelasymmetrie (s.vorher) durchgeführt.

Machen Sie den Fit für die beiden Datensätze.

Erweitern Sie das Programm so dass Sie Ensembletests machen können, d.h. mit den erhaltenen Fitparametern würfeln Sie N-Mal entsprechend verteilte Werte, wiederholen den Fit, und speichern den log-likelihood Wert ab.

Sind beide Datensätze kompatibel mit der angenommenen Verteilung ?

3. Z0 Fit und Korrelationen

Untersuchen Sie die Effekte der Korrelationen beim Z0 Fit (`bwfitmnce.C`) auf die Fitparameter, Fehler und ihre Korrelationen.

Was ändert sich wenn Sie systematische Effekte auf y bzw $E_{CM} = x$ einschalten?

Die Messung von E_{CM} hat neben einer voll korrelierten Unsicherheit von 7 GeV noch einen unabhängigen Fehler (Punkt-zu-Punkt) von 5 GeV.

Führen Sie einen entsprechenden Term in die Kovarianzmatrix ein, wie ändern sich die Ergebnisse?

9.7 Hypothesen

1. Hypothesen Tests I

Folgende Tabelle zeigt die Gewinner in Pferderennen sortiert nach der Startbox.

Startbox	1	2	3	4	5	6	7	8
Gewinner	29	19	18	25	17	10	15	11

Teste die Hypothese, daß die Startbox *keinen* Einfluß auf die Gewinnwahrscheinlichkeit hat mit einem χ^2 Test. Definiere eine Konfidenz von z.B. 95% oder 99% vor Durchführung des Tests.

2. Hypothesen Tests II

Geladene Teilchen, die durch ein Gasvolumen fliegen, erzeugen ionisierte Ladungen, deren mittlere erzeugte Menge vom Teilchentype abhängen. Nehme an, daß eine Teststatistik t anhängig von der Ionisationmessung erzeugt wurde, die einer Gauß-Verteilung mit Mittelwert 0 für Elektronen und Mittelwert 2 für sog. Pionen folgt. Die Standardabweichung für beide Verteilung sei 1. Ein Test für die Selektion von Elektronen verlangt $t < 1$.

- Wie groß ist the Elektron-Selektionseffizienz, d.h. wie groß ist die Wahrscheinlichkeit ein Teilchen zu akzeptieren, wenn es ein Elektron ist ?
- Wie groß ist die Wahrscheinlichkeit ein Pion als Elektron zu selektieren ?
- Nehme an, ein Datensatz besteht aus 99% Pionen und zu 1% aus Elektronen. Wie groß ist die

Reinheit der Elektronen-Selektion mit $t < 1$?

3. Testen von Hypothesen

- Ist es signifikant wenn in einer Untersuchung 10 Patienten auf ein Medikament positiv reagieren und in der Kontrollgruppe nur 7 Patienten?
- Wie signifikant ist es wenn in der Umgebung eines AKW 4 Patienten an einer Krankheit leiden, wenn man im Mittel nur 2 Fälle erwartet? Hier muss man berücksichtigen dass es mehr als ein AKW gibt.
- Kann man ein Modell (z.b. ein Higgs mit Masse < 110 GeV) ausschließen wenn man 10 Untergrund Ereignisse erwartet, das Modell weitere 5 Ereignisse vorhersagt und man 11 in den Daten sieht? (Was ist wenn man nur 9 sieht?)
- Kann man eine neue Entdeckung verkünden wenn man vom Untergrund 10 Ereignisse erwartet aber in den Daten 15 sieht?

9.8 TMVA

1. TMVA I

TMVA ist ein Toolkit für die Multivariate Daten Analyse integriert in ROOT.

- [Die TMVA Projekt-Homepage](#)
- [die TMVA Wiki Seite](#)

TMVA bietet die Möglichkeit verschiedene Algorithmen anzuwenden und komfortabel miteinander zu vergleichen, wie z.B.:

- Rechteckige Schnitt-Optimierung
- Künstliche Neuronale Netzwerke
- Entscheidungsbäume
- etc..

Anleitung für ein kurzes TMVA Tutorial: Trainieren Sie automatisch mit verschiedenen Algorithmen (kann ja nach Algorithmus etwas länger dauern):

```
mkdir tmva  
cd tmva
```

```
root -l \${ROOTSYS}/tmva/test/TMVAClassification.C\ ("Cuts,Likelihood,Fisher"\)
```

Nach einer kurzen Trainingsphase erscheint ein kleines Fenster mit verschiedenen Menüs. Benutzen Sie dieses für die weiteren Aufgaben:

- Welche Eingabevariablen sind miteinander korreliert ?
- Versuchen sie die verschiedenen Algorithmen miteinander zu vergleichen. Welcher Algorithmus zeigt die beste Leistung und warum ?
- Welchen Arbeitspunkt (Signal Effizienz gegen Untergrund-Unterdrückung) würden sie wählen ?

Alle Ergebnisse sind in der Datei TMVA.root gespeichert. Betrachten sie diese mit:

```
root TMVA.root
```

```
TBrowser b
```

Dies öffnet ein Browser Fenster in dem Sie auf der linken Fensterhälfte auf “ROOT files” klicken und dann auf der rechten Seite per Doppelklick durch die Datei “TMVA.root” navigieren können.

Alternativ können Sie auch mit dem TMVAGUI navigieren, falls die Datei “TMVA.root” existiert:

```
root -l \${ROOTSYS}/tmva/test/TMVAGui.C
```

2. TMVA II

Erweitern Sie vorherigen TMVA Übung, indem sie zusätzlich ein Neuronales Netz und ein Boosted Decision Tree trainieren. Benutzen sie hierzu: MLP (Multi Layer Perceptron ist eine Implementation eines Neuronalen Netzes) und BDT:

```
root -l \${ROOTSYS}/tmva/test/TMVAClassification.C\ ("Cuts,Likelihood,Fisher,MLP,BDT)
```

- Welcher Algorithmus zeigt nun die beste Leistung ?
- Welche Variable trägt man meistens zur Netzwerkarchitektur bei ?

9.9 Data Mining

1. Data Mining

Achtung: code nicht mehr aktuell !

Jedes Jahr findet ein sog. Data Mining Cup statt (<http://www.data-mining-cup.de>). Bei diesem Wettbewerb soll man z.B. an Hand von Kundendaten, Vorhersagen auf ein bestimmtes Kundenverhalten treffen.

Diese Daten sind ein idealer Kandidat, um Multi-Variate Analysetechniken auszuprobieren.

```
cd ../examples
cp /home/Johannes.Elmsheuser/mining/2002.tar.gz .
tar xzf 2002.tar.gz
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$HOME/TMVA/lib
make
```

Die Aufgabe aus dem Jahr 2002 steht in der Datei `de_Aufgabe_DMC2002.txt`. Es sollen Stromkunden klassifiziert werden, die zu einem anderen Anbieter wechseln oder nicht.

Die verschiedenen Variable der Trainingsdatei `data_dmc2002_train.txt` sind in `de_Merkmale_DMC2002.txt` beschrieben. Die Daten wurden in verschiedene ROOT Datei umgewandelt: `train.root` enthält alle 10000 Datensätze aus `data_dmc2002_train.txt`, während `train_yes.root` 1000 „canceler“ und `train_no.root` die 9000 „nicht-canceler“

enthält.

Machen Sie sich mit den verschiedenen Variablen vertraut, in dem ROOT mit den gesamten Eingabedaten starten: `root train.root` und anschließend mit dem `TBrowser b` die Variablen anzeigen lassen.

Als nächstes können Sie das Makro `plot.C` verwenden, um sich verschiedene Variablen simulant anzuschauen. Fügen Sie weitere Variablen in der Anzeigen von `plot.C` hinzu. Welche Variablen eignen sich voraussichtlich am besten zu Trennung zwischen „canceler“ und „nicht-canceler“ ?

Verwenden sie TMVA, um verschiedene Multi-Variate Classifier, zu trainieren. Verwenden sie zunächst nur die Fisher Methode

```
./TMVAnalysis Fisher
```

Vergleichen sie die verschiedenen Signal und Background Verteilungen mit:

```
root -l ../macros/TMVAGui.C
```

In der Bildschirm-Ausgabe von TMVA werden auch die Classifier nach ihrer Qualität Signal von Untergrund zu trennen aufgeführt. Welches sind die besten Variablen für die Fisher Methode ?

Anschliessend trainieren sie weitere Classifier für Neuronale Netze (MLP - Multi Layer Perceptron) und Boosted Decision Trees (BDT) mit:

```
./TMVAnalysis Fisher MLP BDT
```

Dieses Training dauert je nach CPU Leistung mehrere Minute. Vergleichen sie auch hier die Leis-

tung der verschiedenen Classifier. Welche Variablen eignen sich hier am besten zur Signal und Untergrundtrennung ?

Wenden Sie die trainierten Classifier auf die „echten“ zu klassifizierenden Daten an mit:

```
./TMVApplication
```

Öffnen Sie anschließend die erzeugte Datei `TMVApp.root` und schätzen Sie erfolgte Klassifizierung der Daten mit Hilfe der verschiedenen Algorithmen ein.

9.10 Optimierung

1. Function Sampling mit Markov Chain MC

Experimentieren Sie MCMC für Funktionen, das Beispielprogramm aus der Vorlesung ist `mcmc-fun.C`

2. Fitten mit MCMC

Das Makro `HGamGam.C` erzeugt das gezeigte (low-statistic) $H \rightarrow \gamma\gamma$ Massenspektrum. `mcmchgg.C` ist der zugehörige MCMC Fit.

Testen Sie verschiedene Startwerte für die Parameter, variieren Sie Schrittweiten, etc.

3. Traveling Salesman Problem

`tspm.C` enthält das Programm zur Generierung einer TSP-Städteanordnung und der Optimierung mit MCMC und Simulated Annealing.

Experimentieren Sie mit den verschiedenen Möglichkeiten zur Weg-Sequenz-Änderung, Temperatur, Abkühlgeschwindigkeit, etc.

Eine interessante Variante ist TSP mit Grenzübertritt: nehmen Sie an dass zwischen linker und rechter Hälfte eine Grenze ist, die sehr aufwendig zu passieren ist ($BRD \leftrightarrow DDR$), und man die Zahl der Grenzübertritte möglichst minimieren will. Das kann man dem Programm einfach beibringen indem man noch einen zusätzlichen Kostenfaktor einführt bei Übergang von links nach rechts und v.v., der zur Weglänge hinzugefügt wird.