Teilchenphysik Softwarekurs
–Übungen–

Günter Duckeck

Sommersemester 2020

Inhalt:

- Ein/Ausgabe von Daten
- Graphische Darstellung
- ROOT tuples und eine kleine Analyse
- Fitten von Daten

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

1 Allgemeines

Kursziele:

- Einführung in und Arbeiten mit ROOT
- Einfaches Gerüst für eine Datenanalyse

Folien & Übungen (pdf) im WWW
http://www.etp.physik.uni-muenchen.de/kurs/comp20/

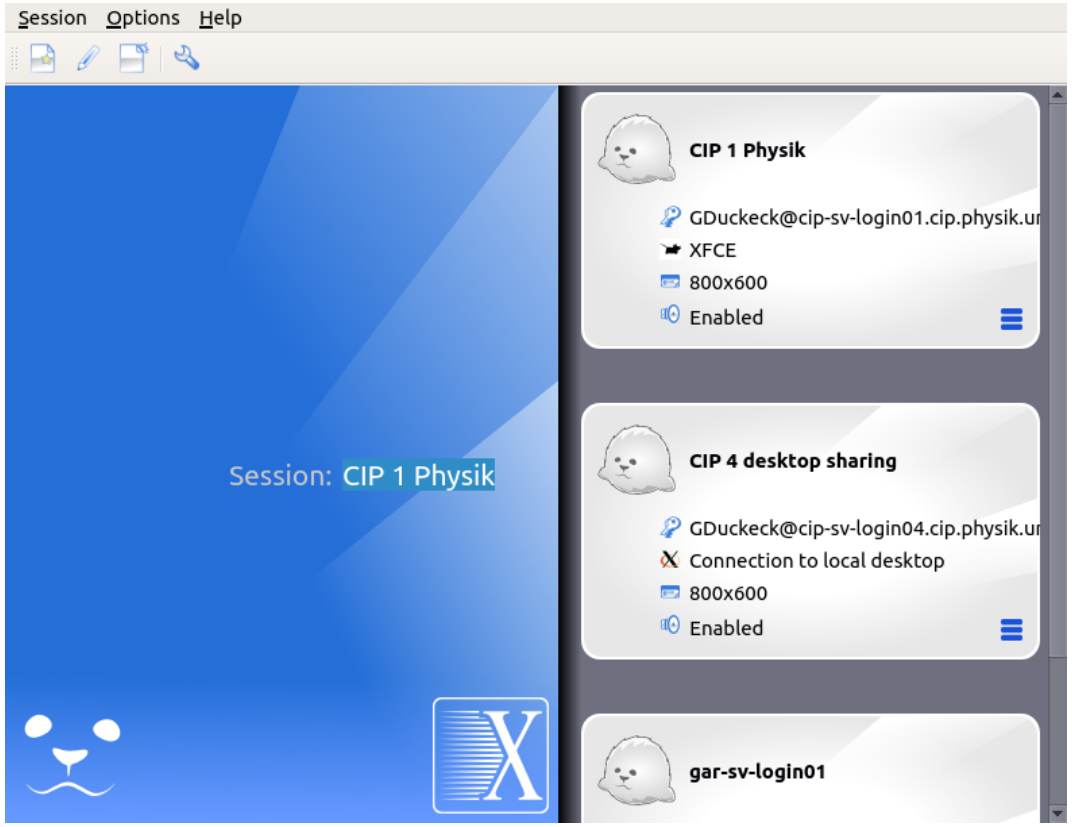
1.1 X2GO für Remote login und Desktop-Sharing

Das Bearbeiten der Programmbeispiele und Übungsaufgaben soll auf den Physik-CIP Rechnern stattfinden. Zugang zum CIP sollte jeder in LMU-Physik haben. Zum Verbindungsaufbau mit grafischer Benutzeroberfläche müssen Sie das Paket **X2GO** auf Ihrem Rechner installieren.

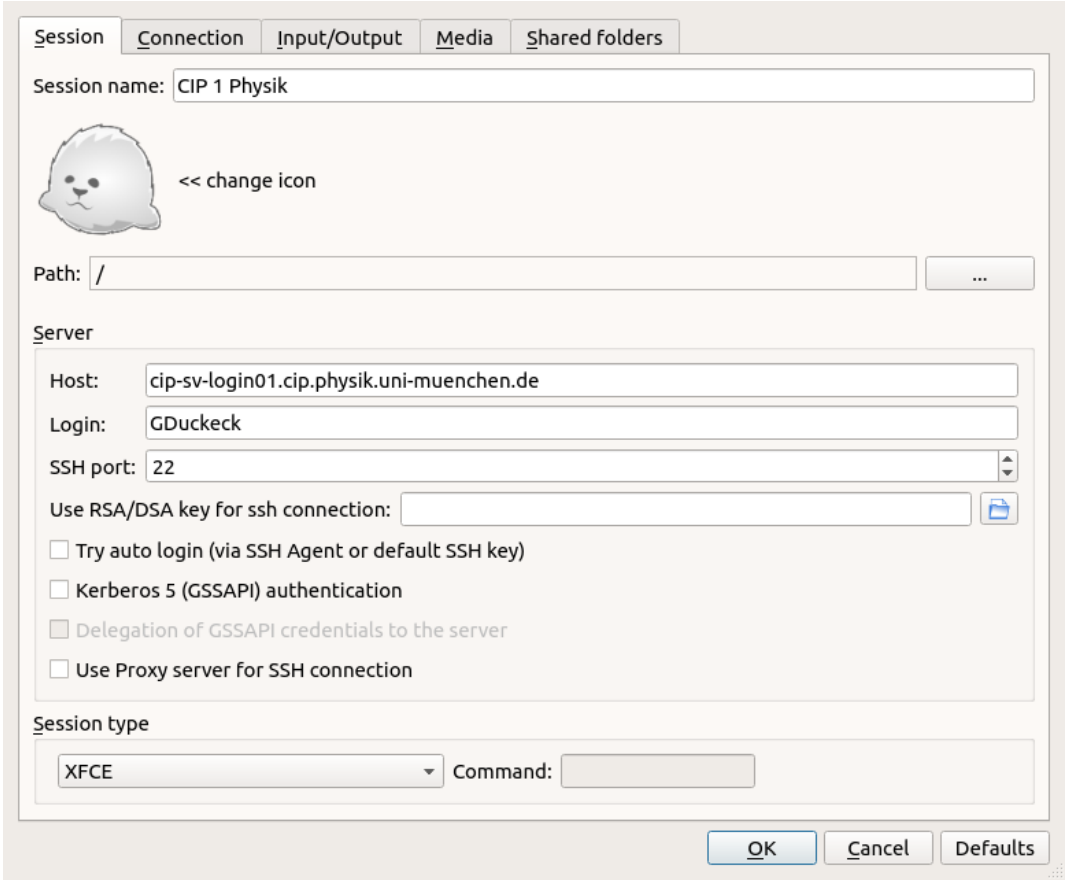
Anweisungen zur Installation siehe
https://www.it.physik.uni-muenchen.de/dienste/netzwerk/rechnerzugriff/zugriff/remote_login/index.html.

X2GO Einstellungen:

Start Fenster


Session Options Help
Session: CIP 1 Physik
CIP 1 Physik
GDuckeck@cip-sv-login01.cip.physik.ur
XFCE
800x600
Enabled
CIP 4 desktop sharing
GDuckeck@cip-sv-login04.cip.physik.ur
Connection to local desktop
800x600
Enabled
gar-sv-login01

Einstellungen

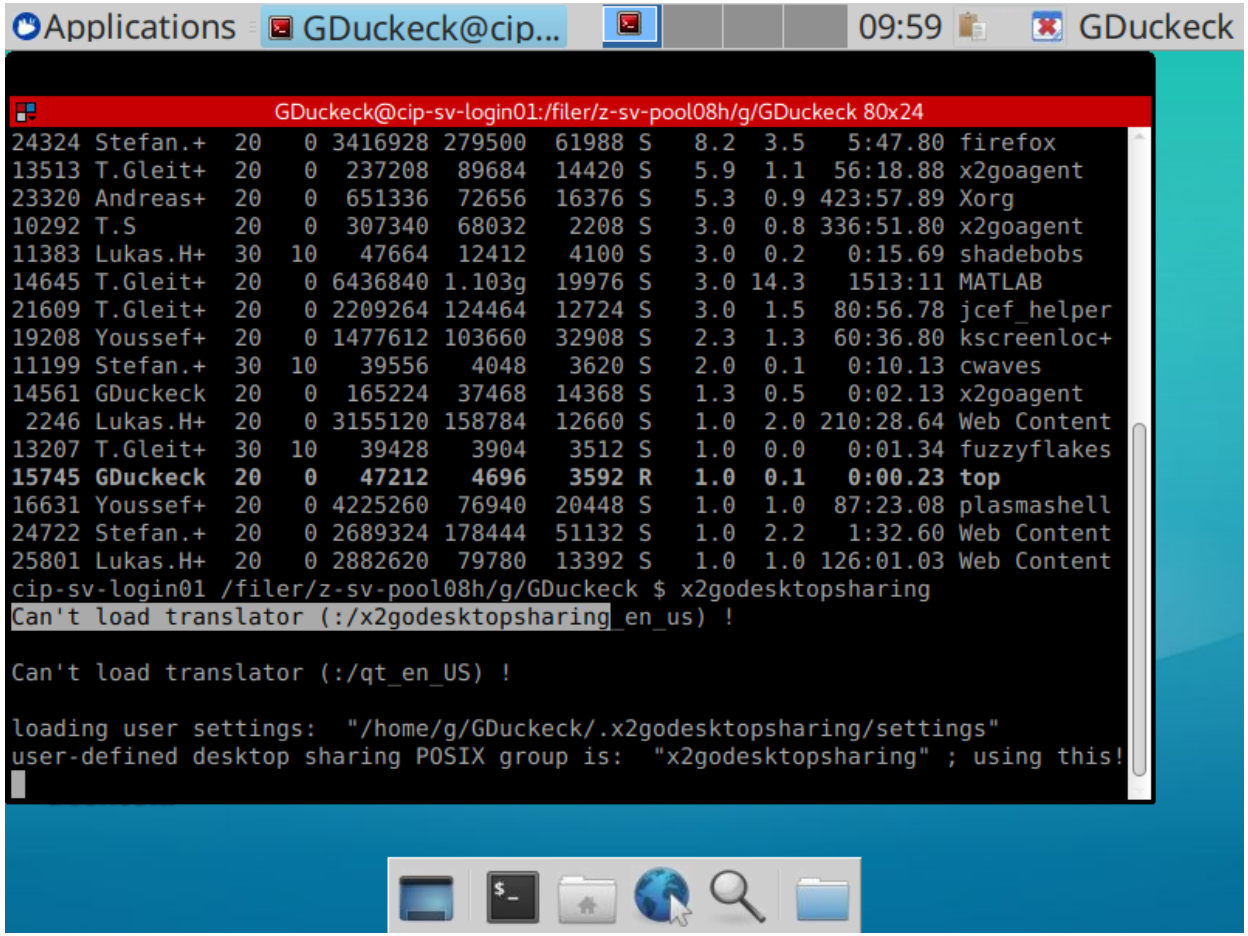


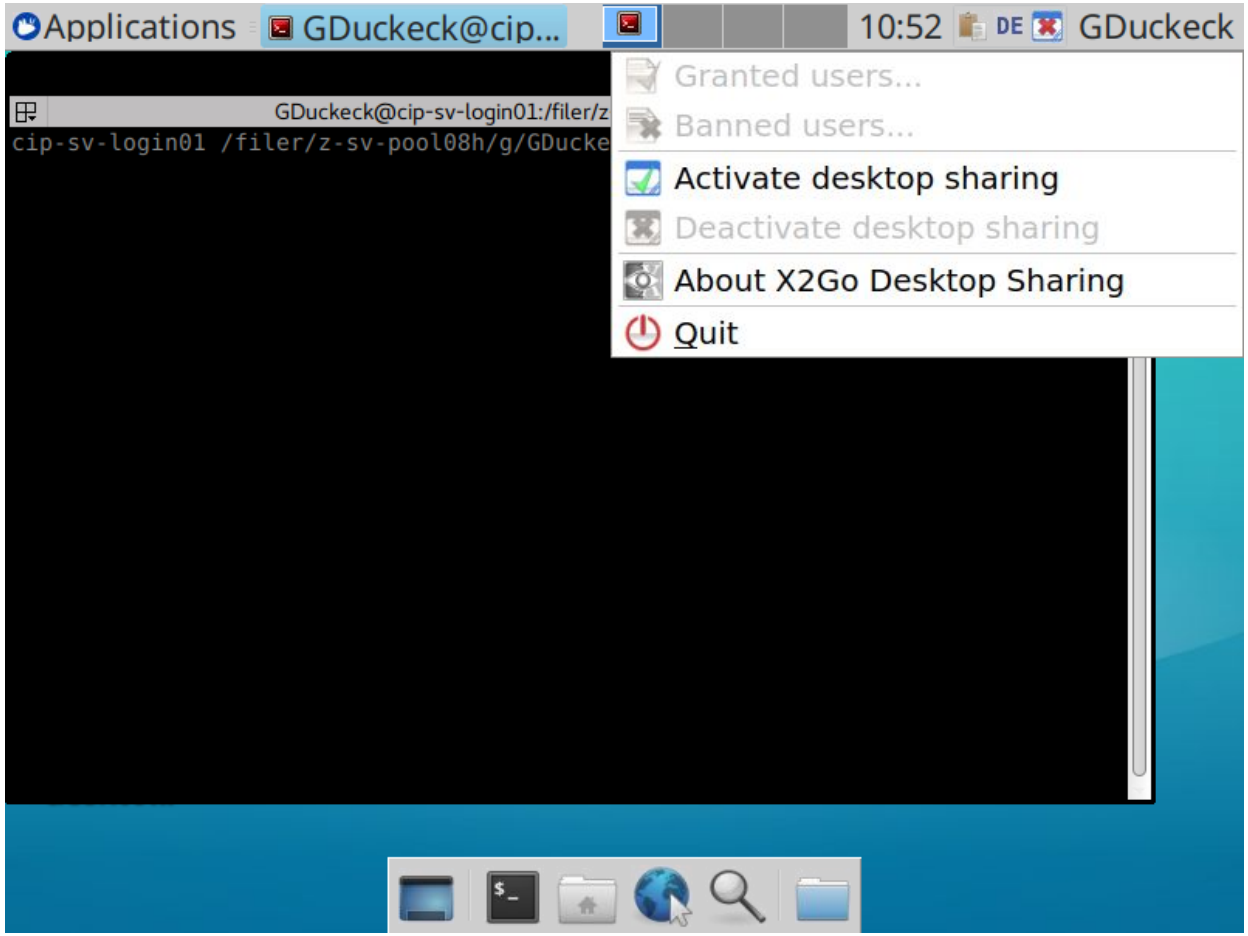
- Feld **Host**: Vollen Hostname eintragen, z.B. `cip-sv-login02.cip.physik.uni-muenchen.de`
- 4 CIP Systeme von aussen zugänglich:
  `cip-sv-login01, cip-sv-login02, cip-sv-login03, cip-sv-login04`
- Feld **Login**: Ihr Login Name
- Feld **Session type**: `XFCE` empfohlen

Session Sharing

Für direkte Hilfe und Unterstützung beim Programmieren kann es erforderlich sein, dass die Betreuer sich von aussen in Ihre X2GO Sitzung einklinken. Dazu sind folgende Schritte nötig:

- Starten Sie Terminal Fenster in Ihrer X2GO Session
- Kommando `x2godesktopsharing` eingeben
- Oben rechts erscheint **Icon mit rotem X** : Mit **rechter Maustaste** anklicken und `Activate desktop sharing` auswählen


Applications
GDuckeck@cip...
09:59
GDuckeck
GDuckeck@cip-sv-login01:/filer/z-sv-pool08h/g/GDuckeck 80x24
24324 Stefan.+ 20 0 3416928 279500 61988 S 8.2 3.5 5:47.80 firefox
13513 T.Gleit+ 20 0 237208 89684 14420 S 5.9 1.1 56:18.88 x2goagent
23320 Andreas+ 20 0 651336 72656 16376 S 5.3 0.9 423:57.89 Xorg
10292 T.S 20 0 307340 68032 2208 S 3.0 0.8 336:51.80 x2goagent
11383 Lukas.H+ 30 10 47664 12412 4100 S 3.0 0.2 0:15.69 shadebobs
14645 T.Gleit+ 20 0 6436840 1.103g 19976 S 3.0 14.3 1513:11 MATLAB
21609 T.Gleit+ 20 0 2209264 124464 12724 S 3.0 1.5 80:56.78 jcef_helper
19208 Youssef+ 20 0 1477612 103660 32908 S 2.3 1.3 60:36.80 kscreenloc+
11199 Stefan.+ 30 10 39556 4048 3620 S 2.0 0.1 0:10.13 cwaves
14561 GDuckeck 20 0 165224 37468 14368 S 1.3 0.5 0:02.13 x2goagent
2246 Lukas.H+ 20 0 3155120 158784 12660 S 1.0 2.0 210:28.64 Web Content
13207 T.Gleit+ 30 10 39428 3904 3512 S 1.0 0.0 0:01.34 fuzzyflakes
15745 GDuckeck 20 0 47212 4696 3592 R 1.0 0.1 0:00.23 top
16631 Youssef+ 20 0 4225260 76940 20448 S 1.0 1.0 87:23.08 plasmashell
24722 Stefan.+ 20 0 2689324 178444 51132 S 1.0 2.2 1:32.60 Web Content
25801 Lukas.H+ 20 0 2882620 79780 13392 S 1.0 1.0 126:01.03 Web Content
cip-sv-login01 /filer/z-sv-pool08h/g/GDuckeck $ x2godesktopsharing
Can't load translator (:/x2godesktopsharing_en_us) !
Can't load translator (:/qt_en_US) !
loading user settings: "/home/g/GDuckeck/.x2godesktopsharing/settings"
user-defined desktop sharing POSIX group is: "x2godesktopsharing" ; using this!


Applications
GDuckeck@cip...
10:52
DE
GDuckeck
GDuckeck@cip-sv-login01:/filer/z
cip-sv-login01 /filer/z-sv-pool08h/g/GDucke
Granted users...
Banned users...
Activate desktop sharing
Deactivate desktop sharing
About X2Go Desktop Sharing
Quit

2 Einführung in Linux und Python/C++

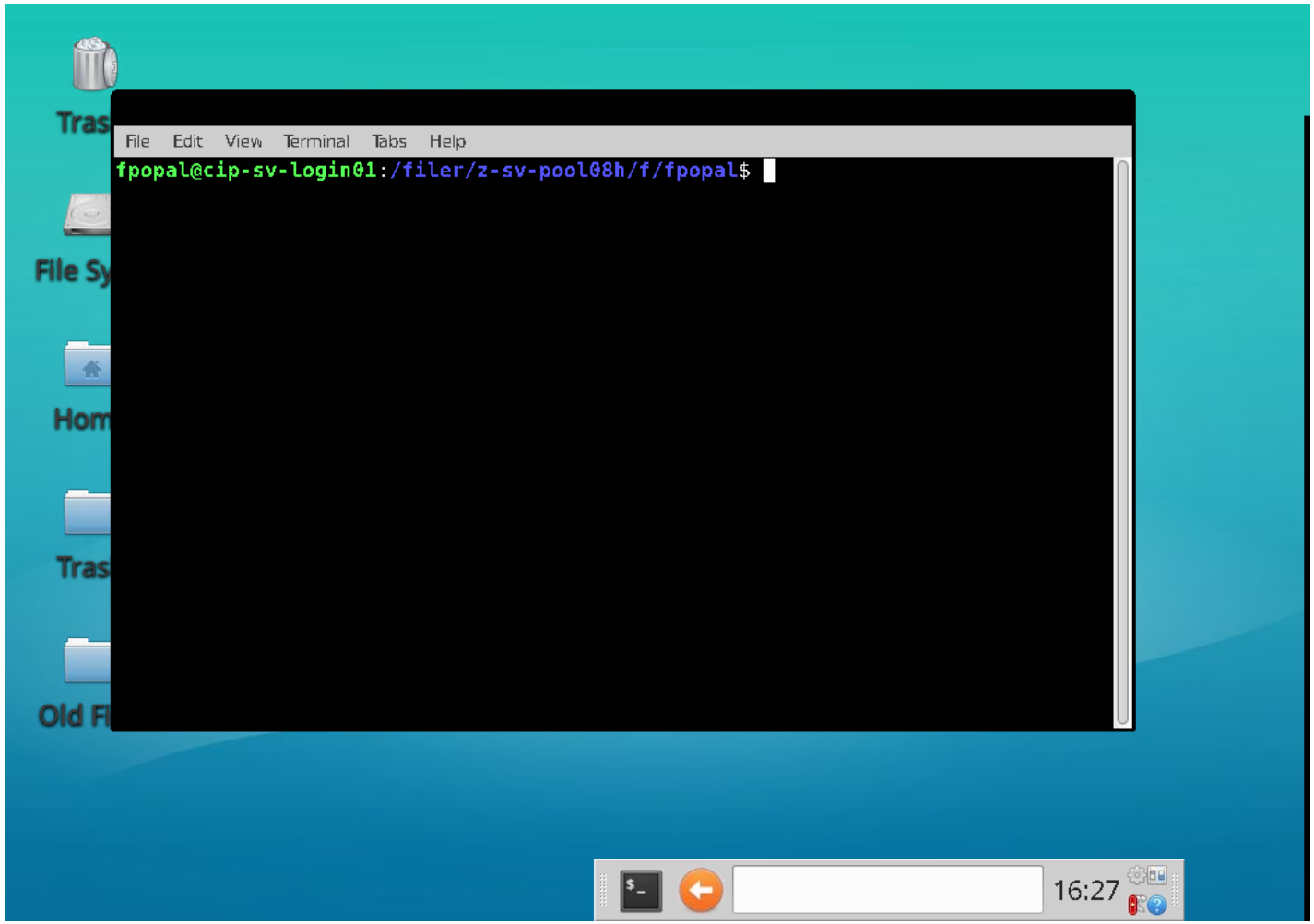
2.1 Links zu Python/C++, Root und Linux Tutorials

- Eine kompakte Einührung in C/C++ gibt es unter: C++ für Physiker
- Und das äquivalente für Python: Python für Physiker
- Ein sehr schönes und ausführliches Tutorial zu ROOT mit Beispielen sowohl in C++ als auch Python gibt es unter: Columbia ROOT Tutorial
- Aktueller ROOT Primer auf der ROOT Homepage

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

- SelfLinux
- UNIX Tutorial for Beginners

2.2 Die Linux X-Benutzeroberfläche mit X2GO



2.3 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 Änderungdatum 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 saemtliche 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

2.4 Rechnerzugriff von aussen

- Grundsätzlich ist Zugang zu den Rechnern via ssh möglich:
  `ssh -XY USERNAME@gar-ws-XX.garching.physik.uni-muenchen.de`
  Allerdings sind die Rechner des ETP Clusters nur innerhalb Münchner Hochschulnetzes direkt zugänglich.
- Zugang von ausserhalb kann man entweder über VPN machen oder über die Gateway-Server
  `gar-sv-login01.garching.physik.uni-muenchen.de,`
  `gar-sv-login02.garching.physik.uni-muenchen.de`
  Weitere Infos
- Grafischer login auch von Windows oder Mac geht via X2GO – Weitere Infos

3 Einführung in ROOT

3.1 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

ROOT ist zentraler Bestandteil der Rekonstruktions- und Analyse Software aller aktuellen Experimente in der Teilchenphysik und wird auch in anderen Bereichen viel verwendet.

ROOT kann sowohl als *Tool-Kit* in größeren, eigenständigen Programmen verwendet werden als auch für interaktive Datenanalyse und Visualisierung.

Wir beschränken uns auf letzteres und einen kleinen Teilbereich der Funktionalität:

- interaktives Arbeiten wahlweise mittels C++ Interpreter `(CINT bzw CLANG)`
- oder Python (pyROOT)
- Direkt von Shell/Kommandozeile
- oder via Jupyter Notebooks

Themenbereiche

- Grundlegende Kommandos
- C++/Python Makros
- Funktionen plotten
- Histogramme und Graphen
- Random numbers
- Fits
- ROOT Trees und einfache Datenanalyse

3.2 ROOT vs Python

Für lange Zeit war ROOT zentrales und unverzichtbares Hilfsmittel für Experimente und Analysen in der Teilchenphysik. Mittlerweile gibt es aber starke Konkurrenz durch Python und seine mächtigen, weitverbreiteten Data-Science Erweiterungen (siehe Python Introduction – M. Ritter).

Im Prinzip gibt es auch für die meisten Komponenten von ROOT Python Bindings, d.h. sie sind von Python aus aufrufbar.

Während bei Belle II Physikanalyen in der Regel komplett im Python Eco-System durchgeführt werden können ist die Situation für ATLAS Analysen eher heterogen:

- ROOT Trees/Files sind weiterhin universeller Standard für Daten I/O in ATLAS
- Einfache Analysen, Fits, Plots lassen sich gut mit den ROOT-Python Bindings in Python abwickeln
- Komplexere Analysen nutzen aber meist existierende Tools/Frameworks in C++, auch wegen Performance
  $\Rightarrow$ Kenntnisse in ROOT-C++ erforderlich

Im nachfolgenden Beispiele zur Verwendung von ROOT mit Python und C++.

3.3 Infos und Links

ROOT :

- **Main ROOT page am CERN**
- **ROOT Klassenindex**
- **ROOT User Guide**
- Ein sehr schönes und ausführliches Tutorial zu ROOT mit Beispielen sowohl in C++ als auch Python gibt es unter: **ROOT Tutorials**
- Weiteres ausführliches Tutorial zur Benutzung von **ROOT (Uni Karlsruhe) (PDF)** bzw. **mit Macros (ZIP)**
- Aktueller ROOT Primer auf der ROOT Homepage

C++ :

- **C++ für Physiker (LMU)**
- **C++ basics for ROOT users**
- C/C++ Referenz Kompakte, übersichtliche Online-Referenz zu C/C++ Funktionen.

Python :

- **Python für Physiker (LMU)**
- Python for Science Schöne Online Referenz mit vielen Physik–Beispielen

3.4 ROOT Installieren

Warnung: Die Installation und Konfiguration von ROOT ist nicht ganz unkompliziert, man muss sich etwas mit Shell Kommandos und Konfiguration auskennen. Insbesondere das Zusammenspiel mit Jupyter und Python ist heikel und erfordert ggf. Installation weiterer Pakete.

Binärpaket installieren (am Beispiel von v6.20.04)

- Root Download Seite: https://root.cern.ch/content/release-62004
- Zum System und C++ Compiler passende Version suchen
- z.B. Ubuntu 18 gcc7.5 : root_v6.20.04.Linux-ubuntu18-x86_64-gcc7.5.tar.gz auswählen und runterladen
- Verzeichnis finden (Downloads?) in dem root_v6.20.04.Linux-ubuntu18-x86_64-gcc7.5.tar.gz abgespeichert wurde
- Evt noch in anderes Verzeichnis verschieben, z.B.:
  `mv Downloads/root_v6.20.04.Linux-ubuntu18-x86_64-gcc7.5.tar.gz .`
- Auspacken:
  `tar -xzf root_v6.20.04.Linux-ubuntu18-x86_64-gcc7.5.tar.gz`
  ROOT wird in Unterverzeicnis ”root” installiert.

- Zur Konfiguration der ROOT Umgebung muss man nun noch:
  `source <aktuelles Verzeichnis>/root/bin/thisroot.sh`
- Anschliessend start mit:
  `root`

Man kann im Prinzip auch das **ROOT source Paket** herunterladen und direkt auf dem jeweiligen Rechner einen Source–Build versuchen. Die korrekte Konfiguration und Installation weiterer Hilfsbibliotheken kann schnell recht komplex werden – empfiehlt sich nur für Leute die sich mit sowas auskennen.

3.5 Arbeiten mit ROOT – C++

3.5.1 ROOT initialisieren und starten

Vor dem ersten Mal:

- Verzeichnis anlegen:
  ```
  mkdir root
  ```
- Initialisierungsfiles kopieren:
  ```
  cp /project/etpsw/Common/root/rootlogon.C .
  ```
-

ROOT starten:

```
// Pfade setzen (evt.  in .bashrc kopieren)
module load root/6.20.04
cd root
root
```

ROOT beenden:

```
.q
```

3.5.2 C++ Operationen – ROOT als Taschenrechner

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

3.5.3 ROOT Datentypen

ROOT benutzt i.W. die Standard-C++ Datentypen:

C++	FORTRAN	Bytes	Bytes	
char	CHARACTER*1	1		1
int	INTEGER	2/4	4	
long		4/8	8	
float	REAL*4	4	4	
double	REAL*8	8	8	

3.5.4 ROOT – einfache interaktive Kommandos

```
root [0] pow(5,2)                                                    1
(int)25                                                              2
root [1] 25.1*3.5                                                    3
(double)8.78500000000000085e+01                                      4
root [2] pow(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)1.00000000000000000e+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
```

3.5.5 ROOT Klassen

ROOT stellt den Anwendern eine riesige Klassenbibliothek zur Verfügung für *Funktionen, Histogramme, Zufallszahlen, Statistik, I/O, etc*.

Arbeiten mit ROOT heisst in der Praxis, dass man Objekte der jeweiligen ROOT Klassen erzeugt und dann Methoden dieser Objekte aufruft.

Einfaches Beispiel:

```
{
  // book histo,
  // arguments: tag, title, N-channels, xlow, x-high
  TH1F myhist("h1","Gauss Random Numbers",100,-5.,5.);
  // random generator object
  TRandom rng;
  for ( int i = 0; i<100000; i++ ) {
    double xrnd = rng.Gaus(); // Gaussian distributed Random number
    myhist.Fill( xrnd ); // Fill random number in histogram
  }
  myhist.DrawClone(); // Draw Histogramm
```

```
}                                                                    12
```

3.5.6 Schleifen

Auch einfache Schleifen kann man interaktiv ausführen ...

```
root [40] double s = 1;                                              1
root [41] for ( int i=1; i<70; i++ ) s *= i        // Fakultaet      2
root [42] s                                                          3
(double) 1.711224524281412 97e+98                                    4
root [57] TMath::Gamma(70)    // Dasselbe mit Gamma Fkt              5
(double) 1.7112245e+98                                               6
```

3.5.7 Macros

Aufruf/Ausführen von C++ Code in externen Files (=Macros):

```
// file fak.C                                                    1
double fak( Int_t n = 1 )                                       2
{                                                               3
  double t = 1;                                                 4
  for ( int 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(99) // dann aufrufen                              13
9.33262e+155                                                    14
```

3.5.8 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)`

```
TH1F fillH1( int n=100000 )                                    1
{                                                              2
  TH1F h2("h2","mytitle",100,0,1);                             3
  for ( int i = 0; i<n; i++ ) {                                4
    h2.Fill(gRandom->Rndm());                                  5
  }                                                            6
  h2.Draw();                                                   7
  return h2; // return histo                                   8
      }                                                        9
```

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 {}:

```
{                                                    1
  int n=100000;                                      2
  TH1F h2("h2","mytitle",100,0,1);                   3
  for ( int i = 0; i<n; i++ ) {                      4
    h2.Fill(gRandom->Rndm());                        5
  }                                                  6
  h2.Draw();                                         7
        }                                            8
```

Äquivalent zur direkten Eingabe dieser Anweisungen auf der Root Kommandozeile, insbesondere bleiben die Variablen erhalten *(global scope)*!

- Ausserdem können (named) Macros auch mit Cling (C++ Interpreter) geladen werden

  `.L myMacro.C`

  ⇒ Datei wird von Cling interpretiert (aber nicht ausgeführt!), es können mehrere Funktionen deklariert werden, Funktionsnamen beliebig, unabhängig von Dateiname.

  Die Funktionen sind anschliessend bekannt und können direkt gerufen werden, z.B.

  `fillH1( 10000 )`

  - Cling ist weitgehend kompatibel mit C++ (im Gegensatz zu CINT, der C++ Interpreter in ROOT Versionen bis v5.x)
  - Standard Root Klassen (TH1F) bekannt, kein `#include <TH1F.h>` nötig.
  - Variablen sind im *local scope* der jeweiligen Funktionen.

- Beim interaktiven Arbeiten macht man oft kleine Änderungen an Makros und möchte diese dann ausprobieren. Das ist in C++ potentiell heikel, weil in C++ Variablen/Funktionen nur einmal deklariert werden können. In aktuellen ROOT versionen ($>=$ v6.20) funktioniert es aber recht gut.

  Falls es doch Probleme gibt kann es helfen vor dem Neu-Laden von Makros explizit ein un-load zu machen:

  `.U myMacro.C`

  Wenn das auch nicht hilft kann man `gROOT->Reset()` versuchen oder gleich Beenden mit `.q` und Neustart.

- **Laden und Kompilieren mit C++ Compiler**
  ist weitere Möglichkeit Programm-Code einzubinden, dazu einfach "+" oder "++" anhängen:

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

  Jetzt wird die Datei mit dem Standard C++ Compiler kompiliert und anschliessend eine *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.

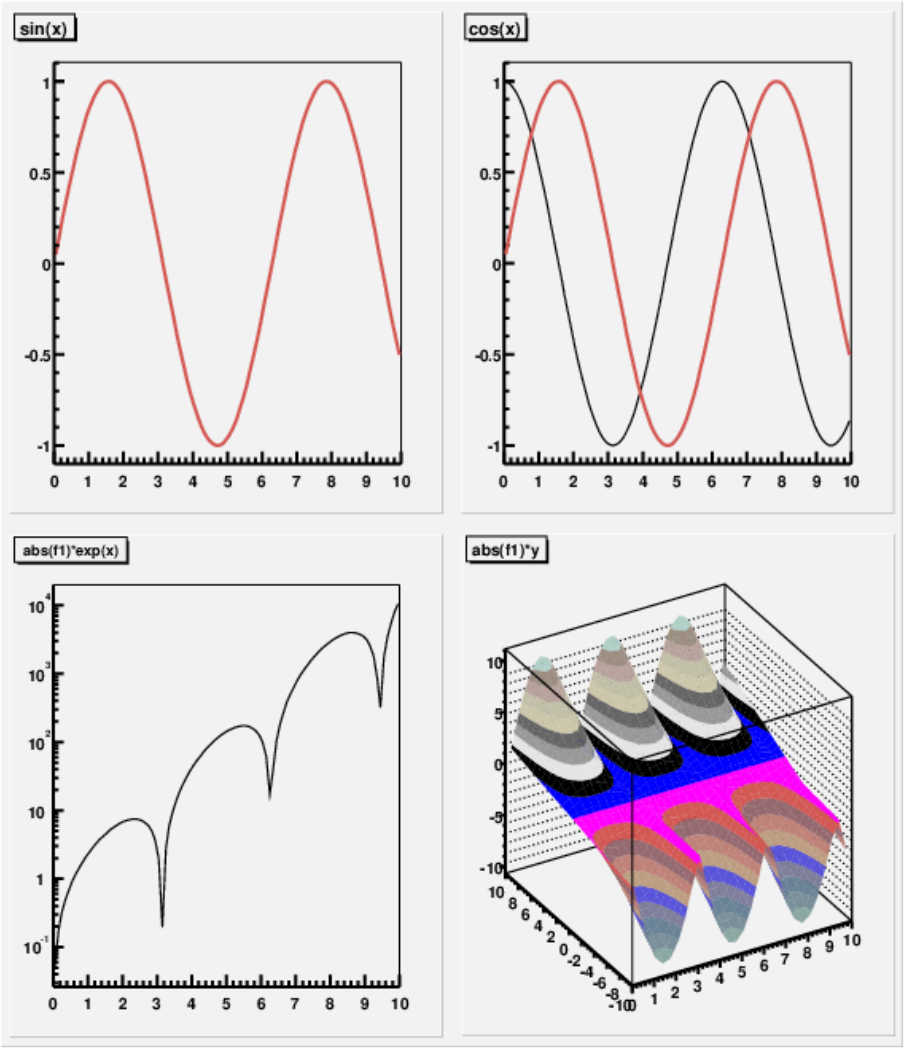
```
#include <TH1F.h>                                                    1
#include <TRandom.h>                                                 2
TH1F fillH1( int n=100000 )                                          3
{                                                                    4
  TH1F h2("h2","mytitle",100,0,1);                                   5
  for ( int i = 0; i<n; i++ ) {                                      6
    h2.Fill(gRandom->Rndm());                                        7
  }                                                                  8
  h2.DrawClone();                                                    9
  return h2; // return histo pointer                                 10
    }                                                                11
```

Bei dieser Methode müssen **alle verwendeten Klassen/Funktionen** über die entsprechenden **Header Dateien** deklariert werden! *(Bei ROOT/Cling sind Standard header für Histogramme, Funktionen, etc. schon bekannt).*

Jetzt gilt üblicher C++ Standard. Der kompilierte Code hat sehr gute Performance.

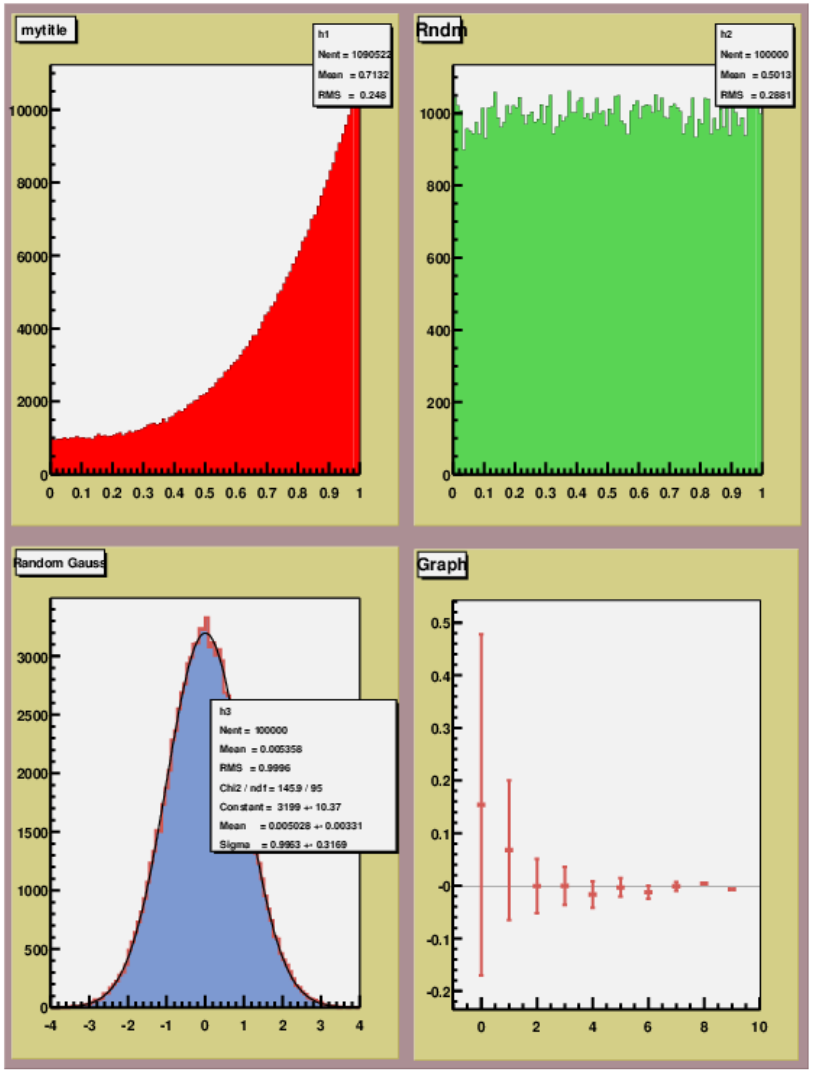
3.5.9 Funktionen

```
root [60] TF1 f1("f1","sin(x)", 0, 10 );
root [61] f1.Draw();
root [68] TF1 f2("f2","cos(x)", 0, 10 );
root [69] f2.Draw();
root [70] f1.Draw("same");
root [76] TF1 f4("f4","abs(f1)*exp(x)", 0, 10 );
root [77] f4.Draw();
root [78] TF2 ff1("f5","abs(f1)*y", 0, 10, -10, 10 );
root [79] ff1.Draw()
```


sin(x)
cos(x)
abs(f1)*exp(x)
abs(f1)*y

3.5.10 Histogramme und Zufallszahlen

```
root [83] TH1F h1("h1","mytitle",100,0,1);
root [84] for ( double x = 0; x<1; x += 1e-6 ) h1.Fill(x,x**3);
root [85] h1.Draw();
root [80] TH1F h2("h2","mytitle",100,0,1);
root [81] for ( int i = 0; i<100000; i++ ) h2.Fill(gRandom->Rndm());
root [82] h2.Draw();
root [87] TH1F h3("h3","Random Gauss",100,-4,4);
root [88] for ( int i = 0; i<100000; i++ ) h3.Fill(gRandom->Gaus());
root [89] h3.Draw();
root [89] h3.Fit("gaus"); // Simple Gauss Fit
```


mytitle
h1
Nent = 1090522
Mean = 0.7132
RMS = 0.248
Rndm
h2
Nent = 100000
Mean = 0.5013
RMS = 0.2881
Random Gauss
h3
Nent = 100000
Mean = 0.005358
RMS = 0.9996
Chi2 / ndf = 145.9 / 95
Constant = 3199 +- 10.37
Mean = 0.005028 +- 0.00331
Sigma = 0.9953 +- 0.3169
Graph

3.5.11 Fitten

```
// file gausf.C
{
  Int_t nit = 10, nrnd, i, j;
  Float_t mean[10], emean[10], xv[10], ex[10];
  TH1F h3("h3","Random Gauss",100,-4,4);
  j =0;
  for ( i =0; i<nit; i++ ) {
    nrnd = 100*pow(2,i);
    xv[i] = i;
    ex[i] = 0.1;
    for ( ; j<nrnd; j++ ) {
      h3.Fill(gRandom->Gaus());
    }
    h3.Fit("gaus","eq");
    h3.Draw();
    TF1 *fit = h3.GetFunction("gaus");
    mean[i] = fit->GetParameter(1);
    emean[i] = fit->GetParError(1);
```

```
    cout << i << " Mean = " << mean[i] << " +- " << emean[i] << endl;
  }
  TCanvas ce("ce", "ce");
  TGraphErrors tg( nit, xv, mean, ex, emean );
  tg.Draw("AP");
}
root [1] .x gausf.C
```

3.5.12 Pointer in C++ und Root

In ROOT wird noch oft mit *Pointern* gearbeitet, d.h. ROOT Objekte und ihre Methoden werden über Pointer initialisiert, angesprochen und zurück gegeben.

Was sind Pointer?

Ein Pointer ist eine Variable, die eine Speicheradresse enthält.

Deklaration `Type * name`, d.h. auch hier Typ-Angabe erforderlich, Unterschied zu *normaler* Variablen–Deklaration ist * vor dem Identifier.

Zuweisung `name = & var`, Address-Zuweisung mittels Adress Operator **&** bei existierenden Objekten oder ...

Zuweisung `name = new Type( ... )`, Address-Zuweisung als Ergebnis von **new ...** bei neu angelegten Objekten.

Verwendung (1) Wert (=Inhalt der Pointer Variablen) ist Adresse

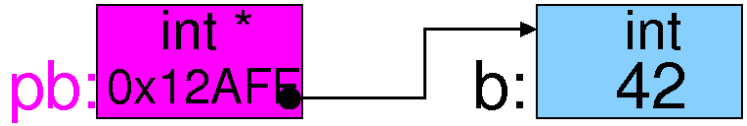
Verwendung (2) **De–Referenzieren**, d.h. Auslesen des Wertes an der Adresse mittels * vor Namen.

Member-Variablen/Methoden Bei Pointern auf Objekte erfolgt der Zugriff auf Member-Variablen/Methoden mit spezieller Syntax `->`, z.B. `histoptr->Fill()` statt `histo.Fill()`

```
int b = 42;                                                                    1
int *pb;    // pb deklariert als pointer auf int                               2
pb = &b; // & ist Adress operator, liefert Adresse von b                       3
cout << b << endl;                                                             4
cout << pb << endl; // Kryptische Adresse                                      5
cout << *pb << endl; // De-referenzieren: *pb == b                             6
double x = 7.256;                                                              7
double *px = &x; // px deklariert als pointer auf double                       8
px = &b; // illegal pointer auf double kann nicht Adresse von int nehmen       9
```

Einfache Variable: Name ist “Label” mit dem Inhalt der Variablen angesprochen wird:

Pointer: Auch Variable, aber Inhalt ist Adresse und Typ einer anderen Variablen:



Direkte Variable vs Pointer in ROOT

```
{                                                                                    1
  // histogram variable direct                                                       2
  TH1F h1("h1","mytitle",100,0,1); // h1 Variable vom Type TH1F                      3
  // Object-method call: objectname.method( ... )                                    4
  for ( int i = 0; i<1000000; i++ ) h1.Fill(gRandom->Rndm()); // Fill                 5
  h1.Draw();    // Draw                                                             6
  // histogram variable as pointer to hist, histo object created with new ...       7
  TH1F * h2 = new TH1F("h2","mytitle",100,0,1);                                      8
  // Object-method call: objectpointer->method( ... )                               9
  for ( int i = 0; i<1000000; i++ ) h2->Fill(gRandom->Rndm()); // Fill               10
  h2->Draw(); // Draw                                                               11
  (*h2).Draw(); // altenative call (de-ref pointer).method( ... )                   12
}                                                                                   13
```

Warum Pointer?

Gültigkeitsbereich (Scope und Lifetime) von Variablen:

- Normal (=direkt) angelegte Variablen und Objekte existieren nur innerhalb Funktion bzw. des Blocks { ... } , wo sie definiert wurden.
- Nur mit `new ...` angelegte Variablen/Objekte existieren über Funktionsgrenzen hinweg und können direkt zurückgegeben werden.

Obiges gilt strikt in kompilierten C++ Programmen. Beim interaktiven Arbeiten mit ROOT/Cling-Interpreter bleiben Histogramme und andere ROOT-spezifische Objekte oft über Funktionsaufrufe hinweg erhalten.

```
TH1F Makehist( int n = 100000)                                                    1
{                                                                                 2
  // histogram variable direct                                                    3
  TH1F h1("h1","mytitle",100,0,1); // h1 Variable vom Type TH1F                   4
  // Object-method call: objectname.method( ... )                                5
  for ( int i = 0; i<n; i++ ) h1.Fill(gRandom->Rndm()); // Fill                   6
  return( h1 );                                                                   7
  // Object h1 will be deleted when function ends                                 8
  // returning h1 to caller requires complex copy operations -> discouraged       9
```

```
}                                                                                    10
TH1F * Makehistp( int n = 100000)                                                    11
{                                                                                    12
  // histogram variable direct                                                       13
  TH1F * h1 = new TH1F("h1","mytitle",100,0,1); // h1 Variable vom Type TH1F *       14
  // Object-method call: objectname.method( ... )                                    15
  for ( int i = 0; i<n; i++ ) h1->Fill(gRandom->Rndm()); // Fill                      16
  return( h1 );                                                                      17
  // Object h1 points to will stay beyond function end,                             18
  // -> programmer's responsibility to delete it                                    19
  // returning pointer h1 to caller fine, no extra copy                             20
}                                                                                    21
```

Wesentlich einfacher in Python: Alle Variablen sind quasi Pointer auf Objekte, automatisches Python Memory Management, ...

3.6 Arbeiten mit ROOT – Python

3.6.1 ROOT initialisieren und starten

Vor dem ersten Mal:

- Verzeichnis anlegen:
  ```
  mkdir root
  ```
- Initialisierungsfiles kopieren:
  ```
  cp /project/etpsw/Common/root/rootlogon.C .
  ```
-

Python/Root starten:

```
# Pfade setzen (evt.  in .bashrc kopieren)
module load root/6.20.04
cd root
#
# interaktive Python Umgebung starten
ipython
>>> import ROOT
```

Python beenden:

```
Ctrl-d
```

3.6.2 Python Operationen – Python/ROOT als Taschenrechner

- alle Python Operationen
  - arithmetisch: `+, -, *, /, **`
  - logical: `and, or, !, <, >, ...`
- Python standard math. Funktionen:

  ```
  import math
  sin, cos, tan, atan, log, exp ...
  ```

- weitere ROOT-Funktionen via ROOT.TMath Klassenbibliothek

```
>>> import ROOT
>>> 5**2
25
>>> 25.1*3.5
87.85000000000001
>>> 25.1**3.5
79224.22969296658
>>> sin(2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
```

```
NameError: name 'sin' is not defined
>>> import math
>>> math.sin(2)
0.9092974268256817
>>> math.atan(1)*4
3.141592653589793
>>> math.log(2)
0.6931471805599453
>>> math.log(1)
0.0
>>> math.exp(1)
2.718281828459045
>>> 2<<10
2048
>>> math.atan2(1,1)
0.7853981633974483
```

3.6.3 ROOT Klassen

ROOT stellt den Anwendern eine riesige Klassenbibliothek zur Verfügung für *Funktionen, Histogramme, Zufallszahlen, Statistik, I/O, etc*.

Arbeiten mit ROOT heisst in der Praxis, dass man Objekte der jeweiligen ROOT Klassen erzeugt und dann Methoden dieser Objekte aufruft.

Einfaches Beispiel:

```
import ROOT
# book histo,
# arguments: tag, title, N-channels, xlow, x-high
myhist = ROOT.TH1F("h1","Gauss Random Numbers",100,-5.,5.)
# random generator object
rng = ROOT.TRandom()
# fill histo in loop
for i in range(100000):
    xrnd = rng.Gaus() # Gaussian distributed Random number
    myhist.Fill( xrnd ) # Fill random number in histogram
myhist.Draw() # Draw Histogramm
```

3.6.4 Schleifen

```
>>> s=1.
>>> for i in range(1,70): s *= i # Fakultaet
...
>>> s
1.7112245242814127e+98
>>> ROOT.TMath.Gamma(70) # Root Gamma Fkt
1.7112245242814127e+98
```

3.6.5 Macros

```
# file fak.py
def fak( n = 1 ):
    t = 1.0
    for i in range(1,n+1):
        t *= i
    return(t)
>>> execfile("fak.py") # execute commands in file fak.py
>>> fak(99)
9.33262154439441e+155
```

3.6.6 Macros cont ...

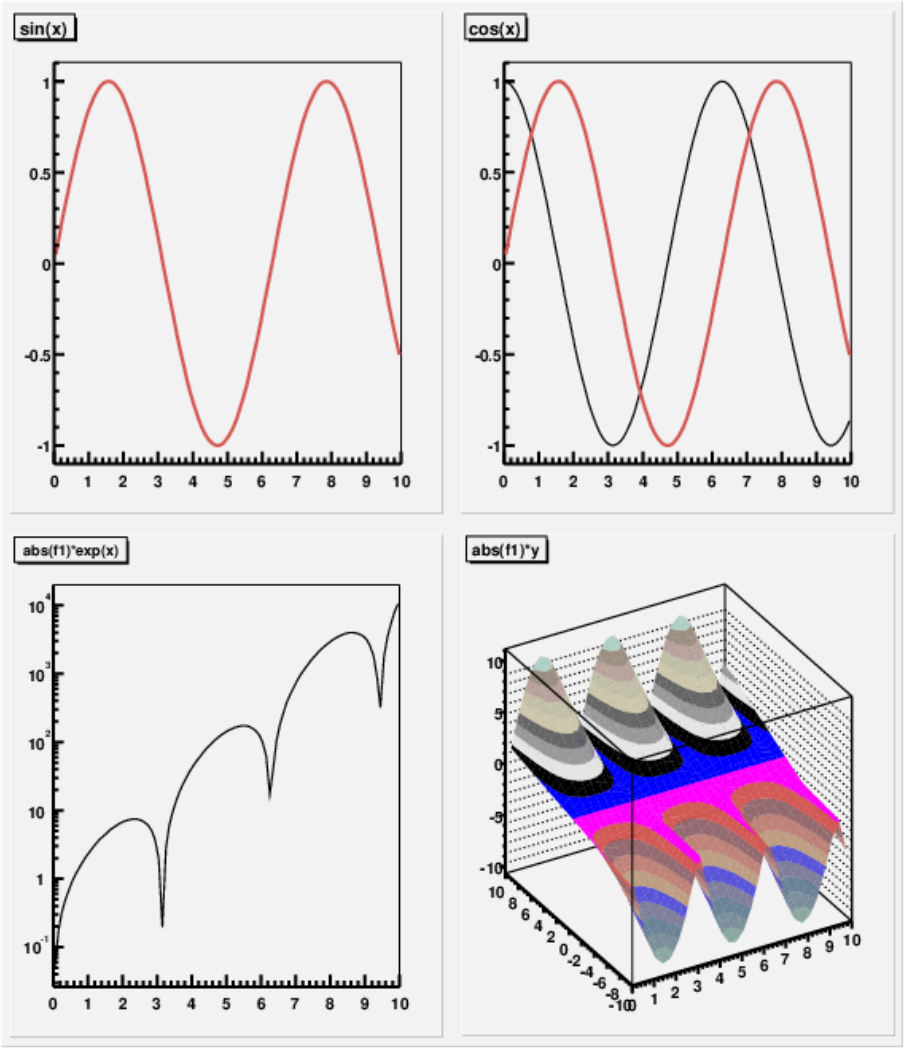
Vorsicht mit Gültigkeitsbereich von Variablen: Alle Variablen, die in der Funktion angelegt werden, sind nach Ausführung wieder verschwunden *(local scope)*

```
def fillH1( n=100000 ):
    h2 = ROOT.TH1F("h2","mytitle",100,0,1)
    for i in range(n):
        h2.Fill(ROOT.gRandom.Rndm())
    h2.Draw()
#
fillH1( 100000 ) # histo wird gefuellt aber nach Draw() wieder weg
```

```
def fillH1( n=100000 ):
    h2 = ROOT.TH1F("h2","mytitle",100,0,1)
    for i in range(n):
        h2.Fill(ROOT.gRandom.Rndm())
    h2.Draw()
    return h2
#
h = fillH1( 100000 ) # histo wird gefuellt und histo Objekt zurueckgegeben
# -> histo bleibt bestehen
```

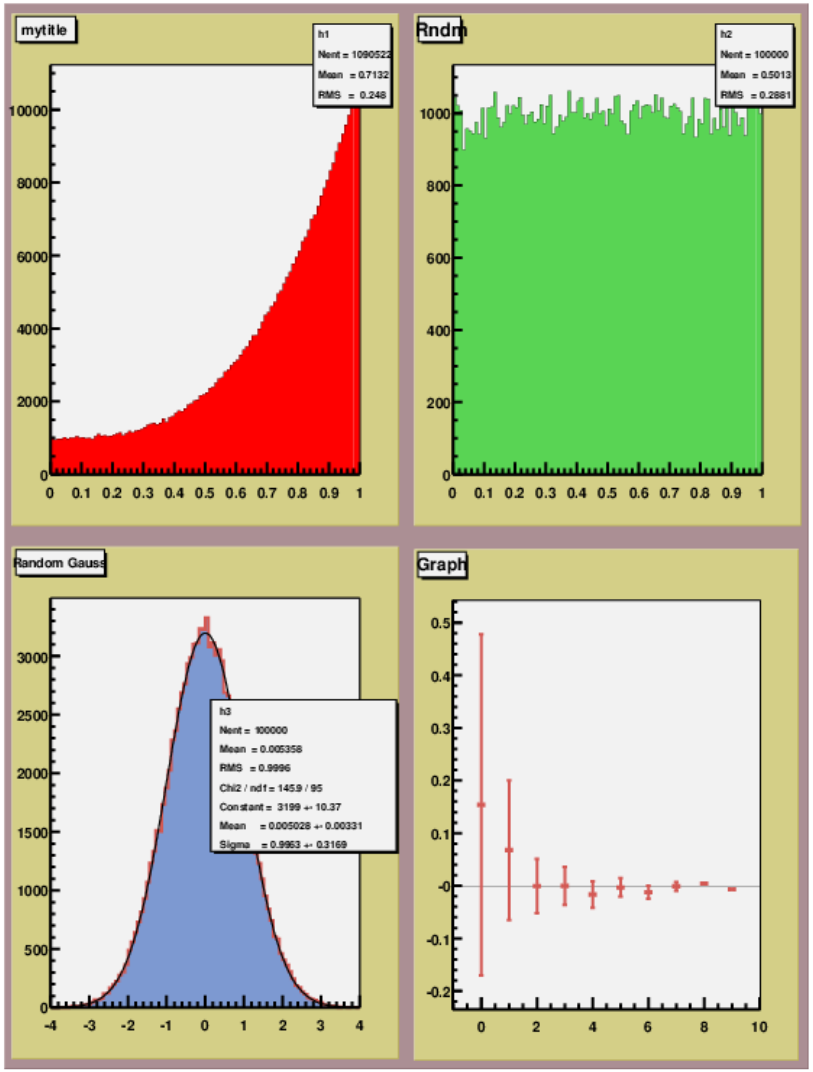
3.6.7 Funktionen

```
>>> f1 = ROOT.TF1("f1","sin(x)", 0, 10 )                                    1
>>> f1.Draw()                                                               2
Info in <TCanvas::MakeDefCanvas>: created default TCanvas with name c1      3
>>> f2 = ROOT.TF1("f2","cos(x)", 0, 10 )                                    4
>>> f2.Draw()                                                               5
Info in <TCanvas::MakeDefCanvas>: created default TCanvas with name c1      6
>>> f1.Draw("same")                                                         7
>>> f4 = ROOT.TF1("f4","abs(f1)*exp(x)", 0, 10 )                            8
>>> f4.Draw()                                                               9
```


sin(x)
cos(x)
abs(f1)*exp(x)
abs(f1)*y

3.6.8 Histogramme und Zufallszahlen

```
>>> h1 = ROOT.TH1F("h1","mytitle",100,0,1)
>>> for i in range(1000000):
...     x = float(i)/1e6
...     r=h1.Fill(x,x**3)
...
>>> h1.Draw()
>>> h2 = ROOT.TH1F("h2","uniform Random",100,0,1)
>>> for i in range(1000000): r=h2.Fill(ROOT.gRandom.Rndm())
...
>>> h2.Draw()
>>> h3 = ROOT.TH1F("h3","Random Gauss",100,-4,4)
>>> for i in range(1000000): r=h3.Fill(ROOT.gRandom.Gaus())
...
>>> h3.Draw()
```


mytitle
h1
Nent = 1090522
Mean = 0.7132
RMS = 0.248
Rndm
h2
Nent = 100000
Mean = 0.5013
RMS = 0.2881
Random Gauss
h3
Nent = 100000
Mean = 0.005358
RMS = 0.9996
Chi2 / ndf = 145.9 / 95
Constant = 3199 +- 10.37
Mean = 0.005028 +- 0.00331
Sigma = 0.9953 +- 0.3169
Graph

3.6.9 Fitten

```
# file pgausf.py
import numpy as np
import ROOT
nit = 10
mean=np.zeros(nit)
emean=np.zeros(nit)
xv=np.zeros(nit)
ex=np.zeros(nit)
h3 = ROOT.TH1F("h3","Random Gauss",100,-4,4)
# produce nit times random Gaus distribution with increasing statistic
# perform each time Gaus fit
# extract fitted mean + error
# plot fitted mean with error
for i in range(nit):
    nrnd = 100*pow(2,i)
    xv[i] = i
    ex[i] = 0.1
    for j in range(nrnd):
```

```
        r=h3.Fill(ROOT.gRandom.Gaus())
    h3.Fit("gaus","eq")
    h3.Draw()
    fit = h3.GetFunction("gaus")
    mean[i] = fit.GetParameter(1)
    emean[i] = fit.GetParError(1)
    print i , " Mean = " , mean[i] , " +- " , emean[i]
ce = ROOT.TCanvas("ce", "ce")
# TGraphErrors needs python array, not just list
tg = ROOT.TGraphErrors( len(xv), xv, mean, ex, emean )
tg.Draw("AP");
#
# in ROOT
#>>> execfile('pgausf.py')
```

3.7 ROOT in Jupyter Notebooks

Eine attraktive Alternative zur Nutzung von ROOT für interaktive Analyse sind *jupyter notebooks*:

- Man startet eine interaktive Python ROOT oder C++/Cling ROOT Session in einem Web Browser Fenster
- Text Output und Plots werden im Notebook angezeigt
- Man kann weitere Kommentare, Beschreibungen, math. Formeln (Latex style) , ..., einfügen

Wird viel verwendet bei Python/Julia/R-basierten Daten-Analysen/Machine-Learning Projekten. Prominentes Beispiel ist die *Ligo gravitational wave analysis* (2017 Nobel prize): Detailiertes Demo Notebook der Datenanalyse

Beispiele zu ROOT Notebooks finden sich auf https://swan.web.cern.ch/content/basic-examples

Start:

```
module load root/6.20.04
```

Mit

```
root --notebook --browser=firefox
```

kann man allgemeine Jupyter Notebook Umgebung starten in der man sowohl *C++ ROOT Notebooks* als auch die üblichen *Python Notebooks* starten kann.

C++ ROOT Notebooks sind etwas speziell in der Handhabung – bedingt durch die **statische Typ Zuordnung** von Variablen in C++ gibt es manchmal Probleme bei der wiederholten Ausführung von Notebook Zellen, was die interaktive Bedienung ziemlich mühsam macht.

Wir beschränken uns deshalb auf *Python Notebooks*, diese kann man sowohl mit `root --notebook` starten als auch direkt als `jupyter notebook`.

Bei Root/PyRoot im Notebook gibt es noch zwei Besonderheiten für die Anzeige von Histogramm/Plot im Notebook:

- Mann muss explizit einen `c = TCanvas(..)` anlegen und anschliessend `c.Draw()` dafür aufrufen damit das Histogramm/Plot im Notebook angezeigt wird.
- Man sollte magic-command `%jsroot on` setzen um interaktiv mit den Plots arbeiten zu können.

Siehe Beispiel Notebook PyROOTNB.ipynb

PyROOTNB Last Checkpoint: 2 hours ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Help Python 2

Code CellToolbar

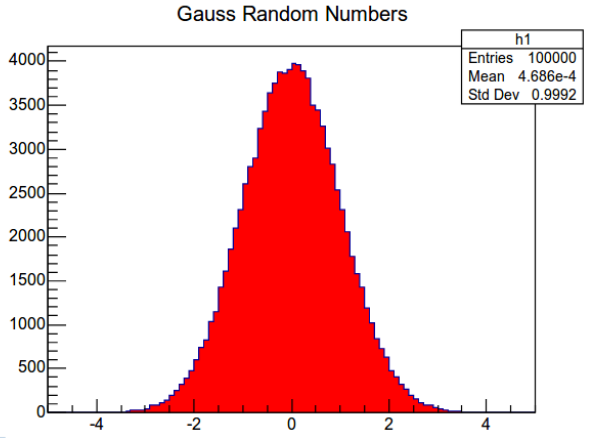
In [1]:
```
# basic setup
import numpy as np
import ROOT
%jsroot on
```

```
Welcome to JupyROOT 6.14/04
```

In [5]:
```
myhist = ROOT.TH1F("h1","Gauss Random Numbers",100,-5.,5.)
# random generator object
rng = ROOT.TRandom()
# fill histo in loop
for i in range(100000):
    xrnd = rng.Gaus() # Gaussian distributed Random number
    myhist.Fill( xrnd ) # Fill random number in histogram

# in Jupyter we need to create explict Canvas ...
c = ROOT.TCanvas("c","myCanvas",500,400)
myhist.Draw() # Draw Histogramm
# ... and call Draw for Canvas
c.Draw()
```

```
Warning in <TROOT::Append>: Replacing existing TH1: h1 (Potential memory leak).
```

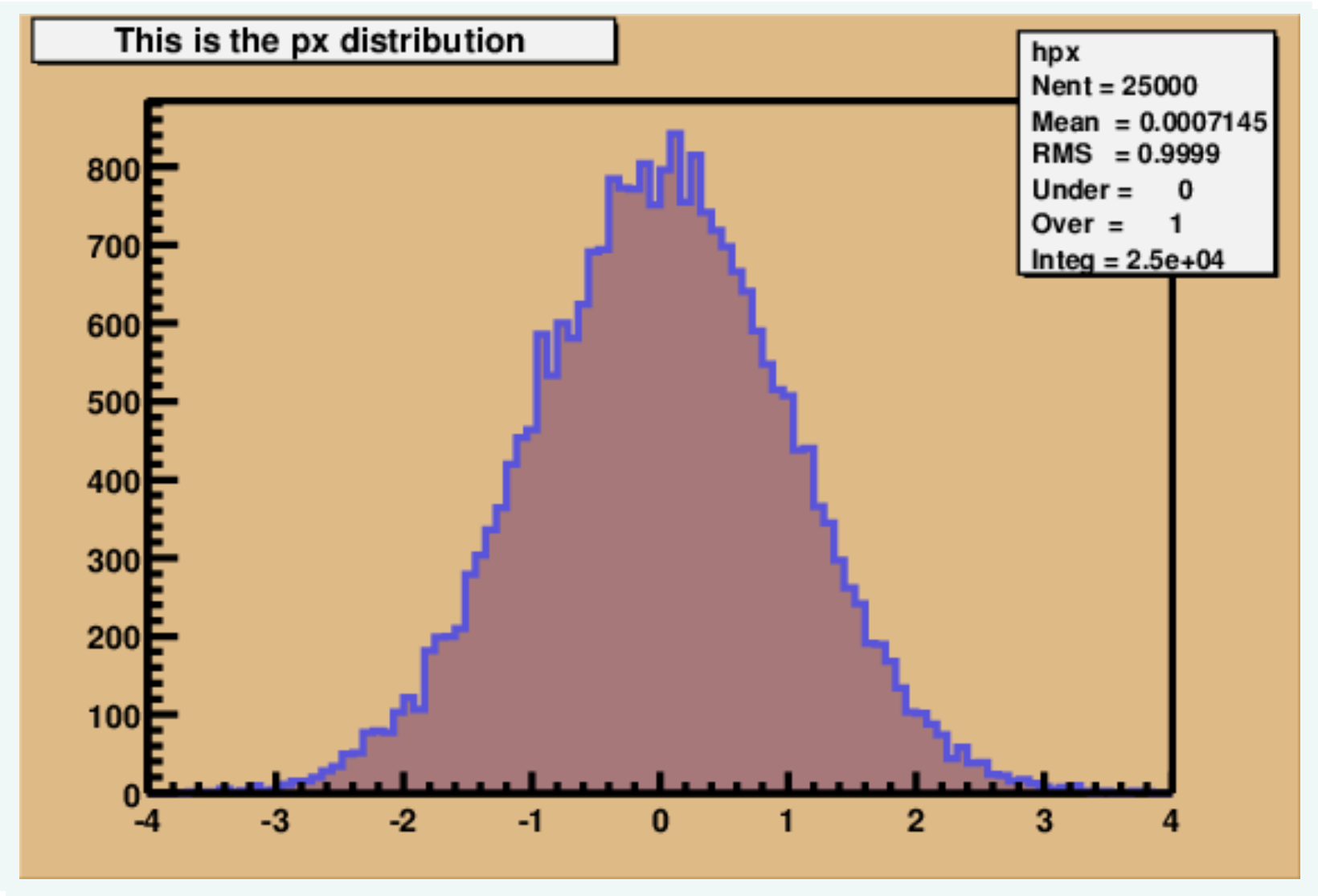


e/jquery-ui.css ...

3.8 Datenvisualisierung

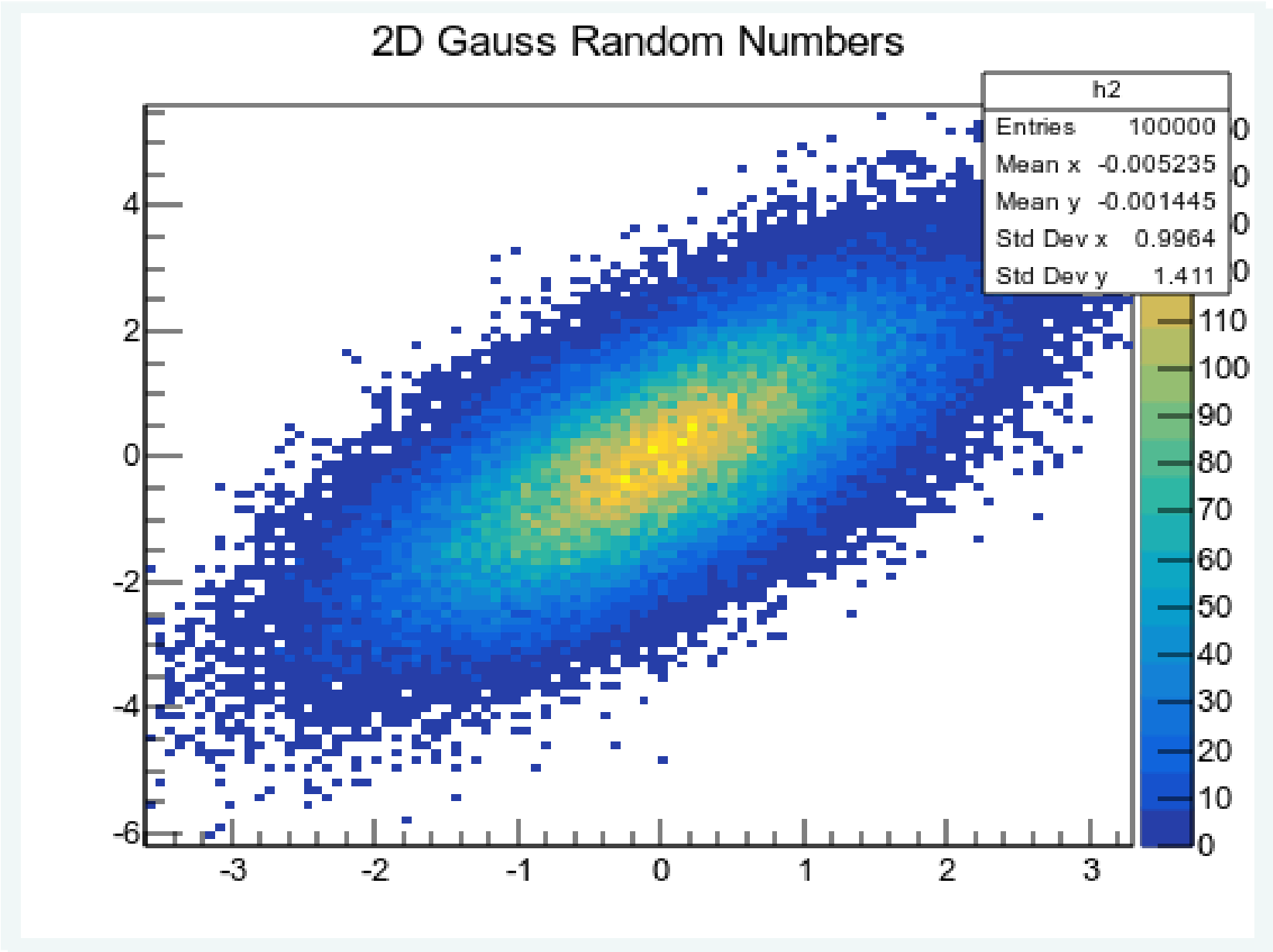
In Naturwissenschaften i.a. zwei Arten von Daten:
Zum Einen die **Häufigkeitsverteilung:** Eine oder mehrere Grössen werden wiederholt gemessen.
Darstellung in ein- oder mehr-dimensionalen **Histogrammen**:

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


This is the px distribution
hpx
Nent = 25000
Mean = 0.0007145
RMS = 0.9999
Under = 0
Over = 1
Integ = 2.5e+04
800
700
600
500
400
300
200
100
0
-4
-3
-2
-1
0
1
2
3
4

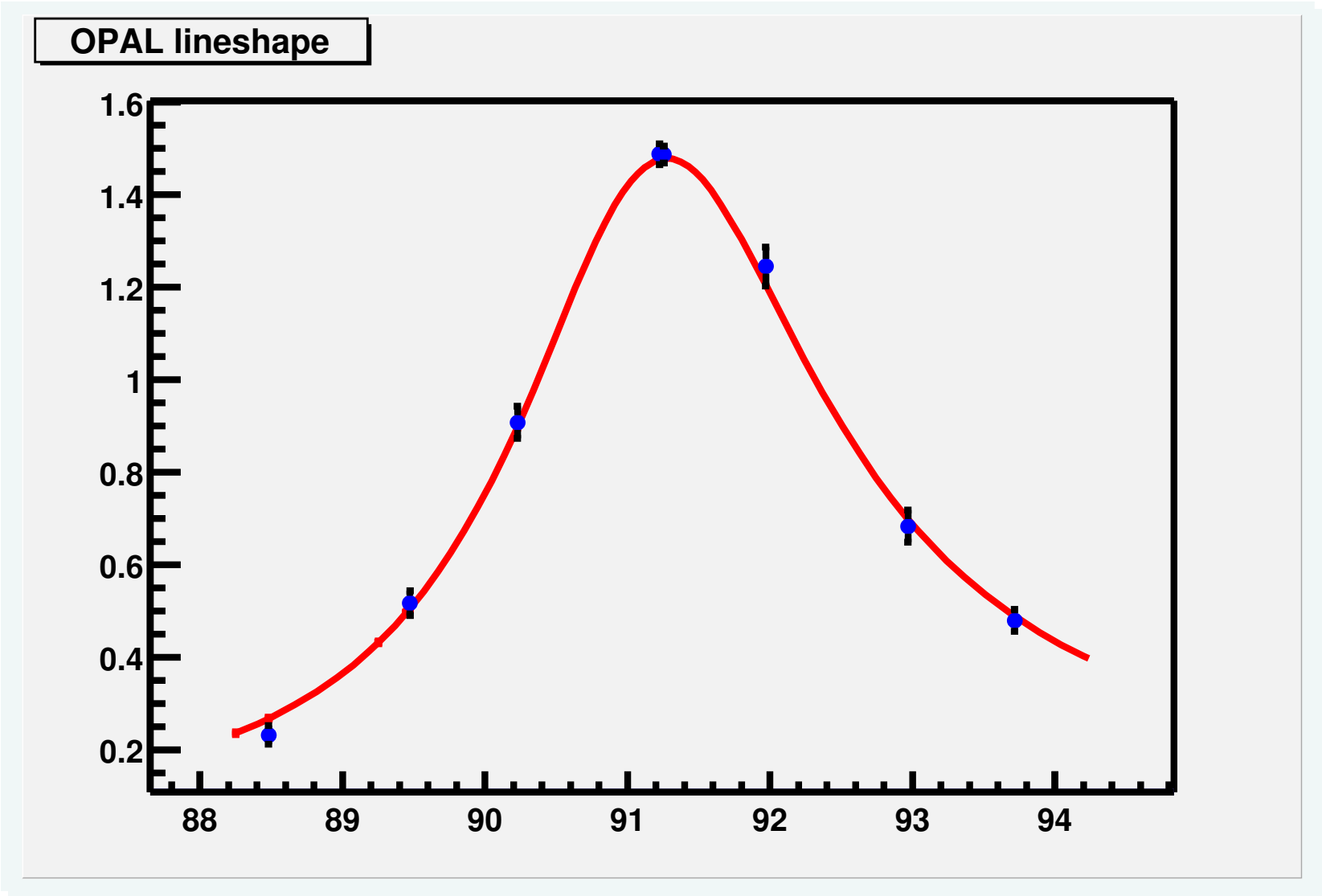
Auch **2 oder mehr-dimensionale Histogramme**:

- Bereich `xlow` bis `xhigh` sowie `ylow` bis `yhigh` unterteilt in `nx * ny` Bereiche.
- `TH2F h2("h2","mytitle",nx, xlow, xhig,ny,ylow,yhigh)`
- Jede Messung `x, y` wird in das Histogram gefüllt:
  `h2.Fill(x,y)`
- Darstellung $\Rightarrow$ Einträge pro Bereich/Rechteck
  viele Varianten: Scatter-Plot, Color-Plot, Box-Plot, ...


2D Gauss Random Numbers
h2
Entries 100000
Mean x -0.005235
Mean y -0.001445
Std Dev x 0.9964
Std Dev y 1.411
4
2
0
-2
-4
-6
-3
-2
-1
0
1
2
3
110
100
90
80
70
60
50
40
30
20
10
0

Zum anderen **(x,y) Wertepaare, ggf. 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
- `tg = TGraphErrors( np, x, y, ex, ey )`
- `x, y, ex, ey` sind jeweils arrays vom Typ `double a[np]` (C++) oder numpy-arrays in Python.


OPAL lineshape
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
88
89
90
91
92
93
94

3.9 Aufgaben

1. **Aufwärmübungen mit ROOT**
   - Initialisieren und Starten Sie ROOT
   - Gehen Sie die Beispiele in diesem Kapitel durch
2. **Histogramm Setup und Zufallszahlen**
   (a) Verwenden Sie das Beispiel Histogramm mit Gaus-Zufallszahlen gefüllt:
   - Variieren Sie Zahl der Einträge, z.B.: 20, 1000, 1e6
   - Was sind jeweils sinnvolle Einstellungen für Zahl Kanäle, untere und obere x-Achsen Grenze?

   (b) Dir ROOT Klasse `TRandom` bietet enthält etliche weitere Funktionen um Zufallszahlen für andere Verteilungen zu erzeugen, z.B. `Poisson, Rndm, Exp, ...`. Probieren Sie einige aus und füllen Sie geeignete Histogramme.
3. **Rohrdaten mit ROOT**
   (a) Lesen Sie die Daten aus `rohr1.dat` in ROOT ein.

   C++ IO

```
ifstream data_file;
data_file.open("rohr1.dat");
double x;
```

```
// book histogram ...
while ( data_file >>x ) // reads next value into x
{ // Fill histo ...
}
```

Python IO

```
import numpy as np
data = np.loadtxt('rohr1.dat') # read all data in numpy array
# book histo ....
for x in data:  # loop over data
# fill histo
```

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

(b) Analog für `rohr2.dat` in ein 2-dim Histogramm ('scatter-plot').

```
TH2F h2("h","mytitle",nx,xlow,xhig,ny,ylow,yhigh);
...
h2.Fill(x,y)
```

C++ IO x,y

```
...
while ( data_file >>x >>y) // reads next value pair into x and y
```

Python IO x,y

```
...
data2 = np.loadtxt('rohr2.dat') # read all data in numpy array
...
for x,y in data2:  # loop over data2 entries
...
```

4. **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)`

4 Datenanalyse – einfach

4.1 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...

4.2 Trees in Root – C++

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

```
root [4] h5000->Scan()
*************************************************************************************************
*    Row   *       Run *     Event *  Ncharged *  Pcharged *    N_ecal *    E_ecal *    E_hcal *   Nmuonen *
*************************************************************************************************
*        0 *      2218 *         1 *        26 * 38.397747 *        27 * 36.205452 * 11.599842 *         0 *
*        1 *      2218 *         2 *        16 * 64.514289 *        28 * 44.801551 * 38.091690 *         0 *
*        2 *      2218 *         3 *        18 * 66.992874 *        25 * 62.047374 * 8.5534896 *         0 *
*        3 *      2218 *         4 *        18 * 48.595024 *        28 * 36.578704 * 21.679370 *         0 *
*        4 *      2218 *         5 *        18 * 61.430469 *        20 * 48.331787 * 27.177442 *         0 *
*        5 *      2218 *         6 *        18 * 54.334602 *        18 * 44.907646 * 24.114786 *         0 *
*        6 *      2218 *         7 *        19 * 35.843792 *        23 * 58.461139 * 22.394620 *         0 *
*        7 *      2218 *         8 *        16 * 41.584270 *        18 * 58.184280 * 3.3126409 *         0 *
*        8 *      2218 *         9 *        16 * 45.887008 *        28 * 49.633556 * 26.143417 *         0 *
*        9 *      2218 *        10 *        25 * 44.548332 *        27 * 55.208030 * 8.8243379 *         0 *
*       10 *      2218 *        11 *        24 * 44.738807 *        38 * 59.603416 * 18.317680 *         2 *
*       11 *      2218 *        12 *        13 * 52.757247 *        26 * 52.511863 * 14.622233 *         0 *
*       12 *      2218 *        13 *        13 * 20.022710 *        30 * 50.100910 * 48.417152 *         0 *
*       13 *      2218 *        14 *        23 * 50.225769 *        28 * 51.006218 * 18.736557 *         0 *
*       14 *      2218 *        15 *        26 * 60.560817 *        30 * 39.966857 * 29.236053 *         0 *
*       15 *      2218 *        16 *        14 * 32.095005 *        26 * 38.665847 * 20.759998 *         0 *
*       16 *      2218 *        17 *        27 * 55.119529 *        33 * 43.566032 * 23.216569 *         0 *
*       17 *      2218 *        18 *        20 * 47.500988 *        28 * 46.652893 * 28.731153 *         0 *
*       18 *      2218 *        19 *        24 * 45.464210 *        27 * 43.797489 * 10.828413 *         0 *
*       19 *      2218 *        20 *        16 * 70.423339 *        13 * 31.889572 * 37.487567 *         0 *
*       20 *      2218 *        21 *        15 * 989.52423 *        19 * 75.841720 * 3.0623478 *         0 *
*       21 *      2218 *        22 *        24 * 62.683868 *        25 * 43.378685 * 22.473934 *         0 *
*       22 *      2218 *        23 *        19 * 45.465820 *        27 * 41.769184 * 33.954277 *         0 *
*       23 *      2218 *        24 *        13 * 34.419353 *         9 * 53.902935 * 9.8699989 *         0 *
*       24 *      2218 *        25 *        18 * 56.245513 *        25 * 67.646148 * 2.1789655 *         0 *
Type <CR> to continue or q to quit ==>
```

4.3 Trees in Root – Python

```
import ROOT
f=ROOT.TFile.Open("http://www.etp.physik.uni-muenchen.de/kurs/comp10/uebungen/Z0-Versuch/data/nt
mytree = ROOT.gROOT.FindObject("h5000") # get access to tree
mytree.Print()
mytree.Scan() # zeigt Variablen
mytree.Draw("Ncharged") # fuellt 1D Histogramm mit Variable Ncharged (automatische Histo Erzeugung)
mytree.Draw("N_ecal:Ncharged") # fuellt 2D Histogramm mit Variable Ncharged vs N_ecal
h1 = ROOT.TH1F("nc","N Tracks", 50, 0, 50) # Buche 1d Histo
mytree.Draw("Ncharged>>nc") # fuellt 1D Histogramm mit Variable Ncharged in gebuchtes Histo
mytree.Draw("Ncharged","E_ecal>10") # fuellt 1D Histogramm mit Variable Ncharged wenn Cut erfuellt ist
```

4.4 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 TTreeViewer:

  - Start direkt von Kommandozeile:
    `TTreeViewer 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 $\Rightarrow$ später mehr

4.5 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 liesst 'branches' aus einer Dateien mit dem Namen 'filename'.

Ein Beispiel ist gegeben in folgender Datei: readfile.C

4.6 Aufgaben

1. Plotten Sie die Verteilungen (1D Histogramme) wichtiger Größen wie `Ncharged`, `E_ecal`, `Pcharged` für die Datensets mit simulierten Ereignissen: ntz0mhmc.root (Quark/Hadronen Zerfälle), ntz0eemc.root (Elektron/Positron), ntz0mmmc.root (Myon), ntz0ttmc.root (Tau)

   Beispiel Notebook zum Prozessieren der Files/Trees und Erzeugen/Plotten der Histogramme: PyROOT-Z0-Ex1.ipynb

2. Machen Sie 2D Histogramme für die drei Kombinationen von `Ncharged`, `E_ecal`, `Pcharged` sowohl für die simulierten Datensets als auch für die Detektor-Daten (ntz0e4.root).

3. Finden Sie einfache Schnitte auf diese Größen um die vier Endzustände zu trennen.

5 Datenanalyse – ausführliches Beispiel Z0-Versuch

5.1 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 $e^+e^- \to Z^0 \to q\bar{q}$ und $e^+e^- \to Z^0 \to \mu^+\mu^-$
- 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` : $e^+e^- \to Z^0 \to q\bar{q}$
    `ntz0mmmc.root` : $e^+e^- \to Z^0 \to \mu^+\mu^-$
    `ntz0eemc.root` : $e^+e^- \to Z^0 \to e^+e^-$
    `ntz0ttmc.root` : $e^+e^- \to Z^0 \to \tau^+\tau^-$

  - 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.

5.2 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

5.3 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 die Erzeugung einer *TSelector* Klasse massgeschneidert für unseren ROOT-Tree:

- `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("ntz0mhmc.root") // Tree File oeffnen
h5000->Process(s1) // Tree Prozessieren
```

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

5.4 Tree Prozessieren/Selektion in Python

In Python lässt sich Prozessieren von Trees und Selektion sehr einfach erledigen:

- Tree öffnen
- Schleife über Einträge
- Tree Variablen werden automatisch an Tree-Objekt angehängt

```
import ROOT
# open file on wbe server
myf = ROOT.TFile.Open("http://www.etp.physik.uni-muenchen.de/kurs/comp10/uebungen/Z0-Versuch/dat
# retrieve tree
mychain = ROOT.gROOT.FindObject( "h5000" )
# N events
nentries = mychain.GetEntriesFast()
print (nentries , " Events in Tree " )
hnct = ROOT.TH1F("hnct","Ncharged",50,0,50)
# loop over tree
i=0
for x in mychain: #   event features attached to x in each iteration
```

```
    i += 1                                                          13
    if i<=20: # print some info for 1st 20 entries                  14
        print x.Event, x.Ncharged, x.E_ecal                         15
    hnct.Fill( x.Ncharged ) # fill Ncharged histo                   16
#                                                                   17
hnct.Draw() # draw hist when done                                   18
```

5.5 Aufgaben

1. Erstellen Sie C++ oder Python Skripte (bzw Notebooks) zum Prozessieren der Trees mit Selektion und Erzeugen und Füllen der Histogramme, analog zur Aufgabe im vorigen Kapitel, d.h. 1D und 2D Histogramme für die Grössen `Ncharged, E_ecal, Pcharged` für die simulierten Datensets.

   Beispiel Notebook zum Prozessieren der Files/Trees und Erzeugen/Plotten der Histogramme: PyROOT-Z0-Ex2.ipynb