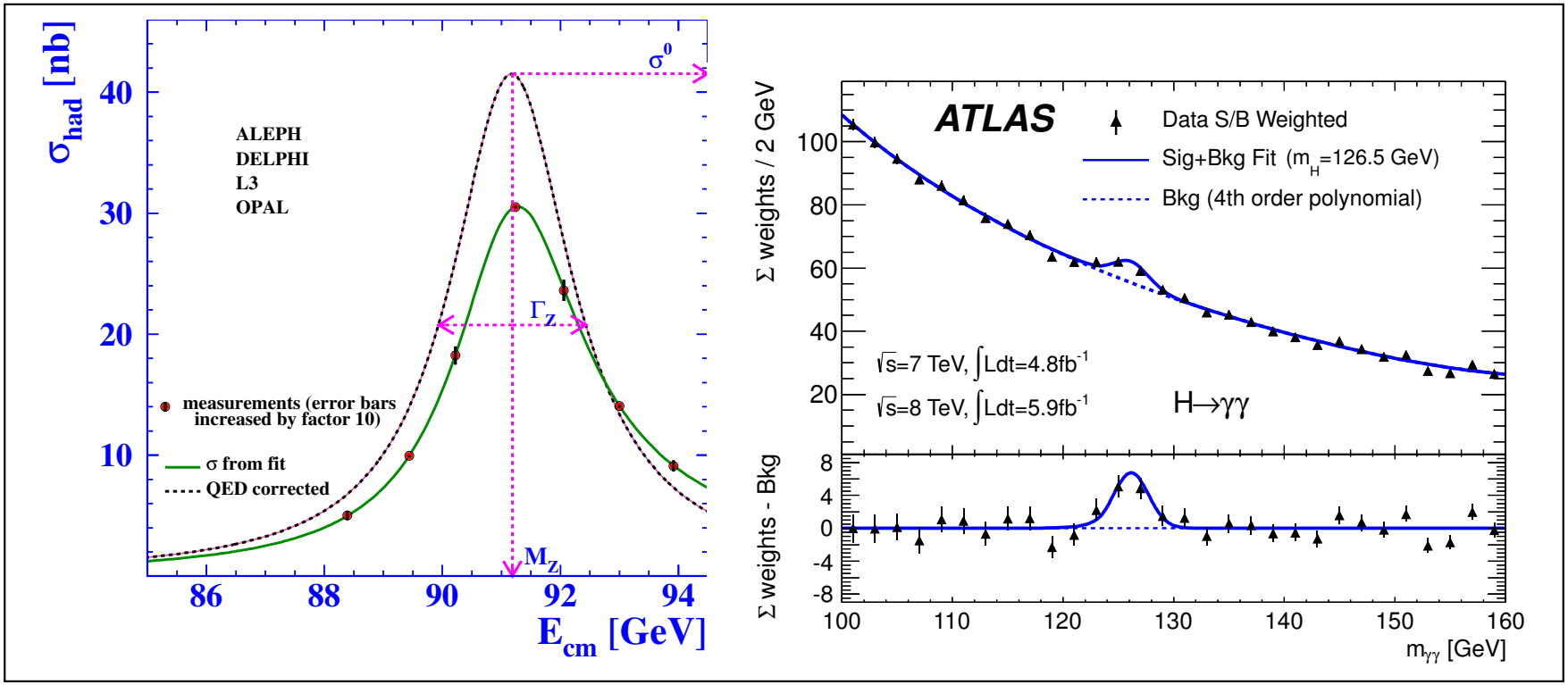
Fits - Schätzung von Parametern

12.04.2018 / Blockkurs: Datenauswertung in der Teilchenphysik

Einführung

Fits von Funktionen an Datenpunkte oder Verteilungen sind Basis für experimentelle Arbeiten in den Naturwissenschaften generell



Einführung

In Kern- und Teilchenphysik werden Fits in allen Phasen eines Experiments verwendet, angefangen von Design und Simulation über Kalibration und Datennahme bis hin zum physikalischen Endresultat einer Messung.

$\rightarrow$ Auf diesen Punkt gehen wir im zweiten Teil heute nochmal genauer ein! Nun zunächst ein paar 'Basics'.

Zwei Arten zugrundeliegender Daten:

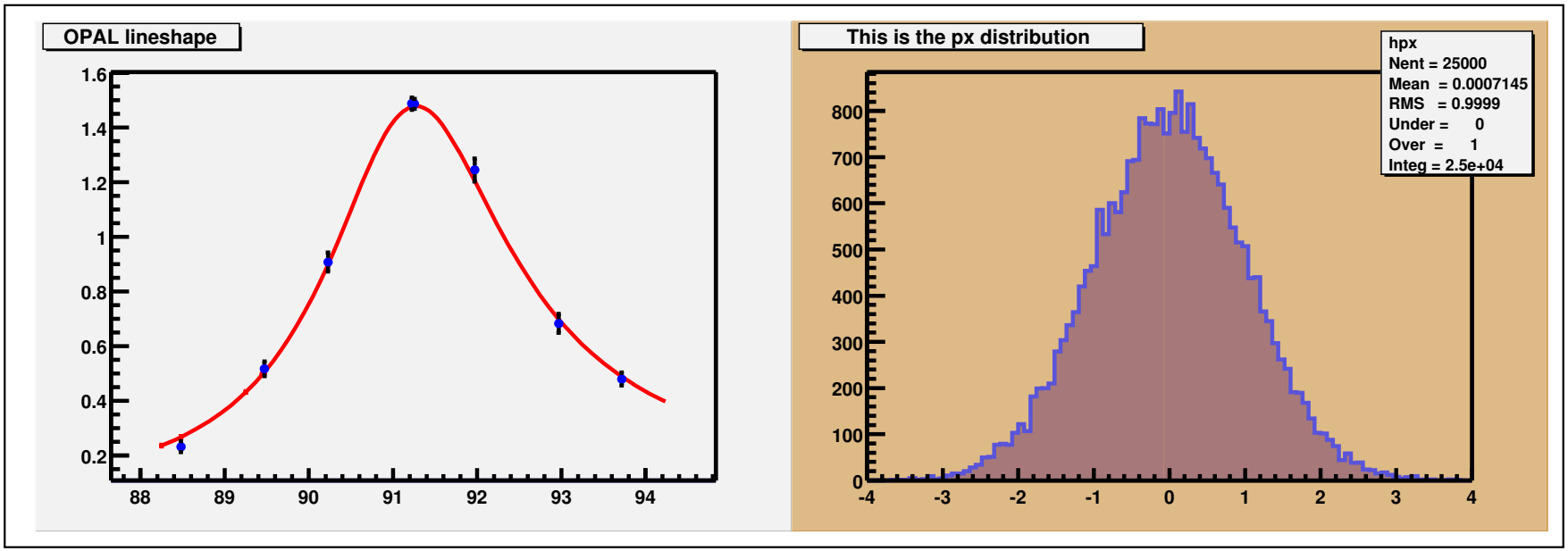
- (x,y) Wertepaare mit Fehlern Δy, für die eine funktionelle Abhängigkeit bestimmt werden soll:

  $y = f(x, \vec{a})$

  Einfachstes Beispiel: Geradenfit: $y = a_0 + a_1 \cdot x$
- Häufigkeitsverteilung $n(x)$. Gesucht ist die zugrundeliegende Wahrscheinlichkeitsverteilung $p(x, \vec{a})$ bzw. ihre Parameter.

Einführung

Typisches Beispiel: Viele Messungen derselben Größe, normalverteilt um Mittelwert, gesucht sind Mittelwert und Streuung.



Im allgemeinen ist man dabei nicht nur an dem Parametersatz $\vec{a}$ interessiert, der die Daten am besten beschreibt. Genauso wichtig sind die Unsicherheiten der Parameter $\Delta\vec{a}$ und ihre Korrelationen *(später)*.

Zwei gängige Verfahren zum Fitten: Methode der kleinsten Quadrate (χ^2 Fit) und Maximum Likelihood

χ^2 Fits

$\rightarrow$ Das gängigste Verfahren, insbesondere bei Fits von Wertepaaren mit normalverteilten Fehlern.

Methode:

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

$\rightarrow$ Finde Parameter $\vec{a}$, die χ^2 minimieren. Für spezielle Fälle (lineare Abhängigkeit der Funktion f von den Parametern $\vec{a}$) analytisch lösbar durch Differenzieren und Lösen der Gleichungen $\frac{\partial \chi^2}{\partial a_k} = 0$

Im allgemeinen Fall muss mit numerischen Verfahren das Minimum bestimmt werden.

χ^2 Fits

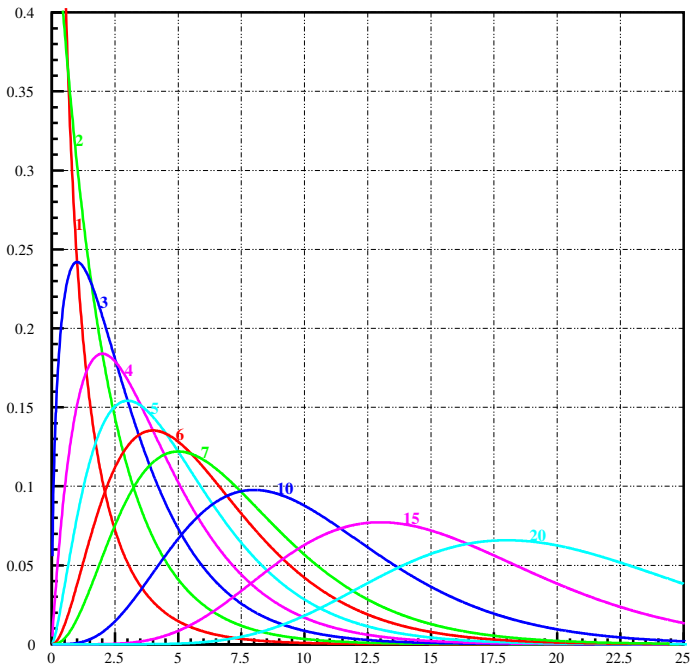
Sehr nützlich ist ein Nebeneffekt des χ^2 Fits:

Der Wert 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:

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

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.



χ^2 Fits

Als Faustregel sollte man sich merken:

$\chi^2/ndf \approx 1$

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.

χ^2 Fits

Wesentlicher Nachteil des χ^2 Fits ist, daß er nur bei normalverteilten Fehlern für die Messpunkte verlässliche Schätzungen der Parameter liefert.

$\rightarrow$ Das ist insbesondere bei Fits von Häufigkeitsverteilungen ein Problem:

- Die einzelnen Werte können nicht direkt benutzt werden sondern müssen in ‘bins’ zusammengefasst werden.
- Diese bins müssen genügend breit sein um jeweils eine vernünftige Zahl (> 5) von Einträgen zu haben, damit man die statistische Unsicherheit als gaussförmig annehmen kann.
- Verlust von Information durch zu breite Bins

Fehlerabschätzung der Parameter

Für ‘anständige’ Funktionen $f(x_i, \vec{a})$ hängt χ^2 am Minimum parabolisch von den a_i ab, d.h

$$\chi^2 - \chi^2_{min} \propto (a_i - a_i^{min})^2$$

Die 1-sigma Fehler der Parameter, Δa_i, entsprechen einer Änderung $\Delta\chi^2 = 1$.

Maximum-Likelihood Fit

Die Probleme des χ^2 fits für nicht-gaussische Fehler oder für Häufigkeitsverteilungen kann man mit einem Maximum-Likelihood Fit vermeiden.

Z.B. für gemessene Werte x_i kann man die Parameter $\vec{a}$ einer Häufigkeitsverteilung $p(x, \vec{a})$ bestimmen aus dem Produkt der Wahrscheinlichkeiten:

$$\mathcal{L}(x_1, x_2, ..., x_n|\vec{a}) \equiv p(x_1, \vec{a}) \cdot p(x_2, \vec{a}) \cdot ... \cdot p(x_N, \vec{a}) = \Pi p(x_i, \vec{a})$$

Die Schätzung $\vec{a}$ ist derjenige Wert, der $\mathcal{L}$ maximiert.

Maximum-Likelihood Fit

Praktischer ist es statt $\mathcal{L}$ den Logarithmus zu benutzen:

$$\ln \mathcal{L} = \sum_{i=1}^{n} \ln p(x_i, \vec{a})$$

$\mathcal{L}$ und $\ln \mathcal{L}$ haben ihr Maximum an der gleichen Stelle.

Einfaches Beispiel: Gauss-Verteilung

$$\mathcal{L}(x_i|\mu,\sigma) = \pi \frac{1}{\sqrt{2\pi}\,\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$$\ln \mathcal{L} = \sum(-\frac{(x-\mu)^2}{2\sigma^2} - \ln(2\pi\sigma^2)/2)$$

Mittelwert durch Differenzieren nach μ:

$$\frac{\partial \mathcal{L}}{\partial \mu} = 0 \rightarrow \frac{\sum(x-\mu)}{n} = 0 \rightarrow \mu =< x >$$

Unbinned Maximum Likelihood Fit

Dieses direkte Fitten einer Häufigkeitsverteilung an die Daten x_i nennt man *Unbinned Maximum Likelihood Fit*.

Im Prinzip ist diese Methode optimal zum Fitten von Häufigkeitsverteilungen

- Die Information wird voll ausgenutzt, keine Verluste durch binning
- Funktioniert auch bei geringer Statistik (wenig Messwerte)

Allerdings gibt es auch ein paar Nachteile:

- 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 ziemlich rechenintensiv
- Keine direkte Visualisierung möglich

Binned Maximum Likelihood Fit

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.
- Die Zahl der Einträge in den einzelnen Bins ist Poisson-verteilt.
  $\ln \mathcal{L}(n_1, n_2, ..., n_n|\vec{a}) = \sum_{bins}(\ln \frac{\mu_i^n e^{-\mu}}{n_i!})$
  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.
  $\rightarrow$ Übungen

Statistische und systematische Fehler

Statistische Fehler, Δ_{stat}, aufgrund der inhärenten Zufälligkeit des Prozesses bzw. der Messung.
Systematische Fehler, Δ_{sys}, durch Unsicherheiten im Verhalten der Messaparatur oder externe Korrekturen.

Beispiel: Messung der Aktivität einer radioaktiven Quelle:

N Ereignisse im Detektor unter Raumwinkel Ω, mit Effizienz ε, in einer Zeit T.

- Δ_{stat}: Unsicherheit in N, für grosse N Gauss-verteilt mit $\sigma = \sqrt{N}$
- Δ_{sys}: Ω, T, ε nicht beliebig genau bekannt.
  $\rightarrow$ korrelierte Unsicherheit bei Wiederholung des Experiments.

Statistische und systematische Fehler

Generell haben Messungen häufig gemeinsame (=korrelierte) systematische Fehler, aber nie ('Primär'–Messungen) gemeinsame statistische Fehler.

I.a.Δ_{sys} schwierig abzuschätzen (1 σ?, Verteilung?)
$\rightarrow$ Experiment–Design: $\Delta_{sys} < \Delta_{stat}$.

Fitten korrelierter Daten

Häufig gibt es Korrelationen zwischen Messwerten, z.B.

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

Dann muss die ‘Kovarianzmatrix’ für den Fit benutzt werden.

Fitten korrelierter Daten

Für n Werte $y_1, y_2, ..., y_n$ beschreibt die $n \times n$ Kovarianzmatrix $\mathcal{C}$ Fehler und Korrelationen der einzelnen Werte:

$$\mathcal{C} = \begin{pmatrix} \sigma^2_{y_1} & cov_{y_1,y_2} & \dots & cov_{y_1,y_n} \\ cov_{y_1,y_2} & \sigma^2_{y_2} & \dots & cov_{y_2,y_n} \\ \dots & \dots & \dots & \dots \\ cov_{y_1,y_n} & cov_{y_2,y_n} & \dots & \sigma^2_{y_n} \end{pmatrix}$$

mit $cov_{y_i,y_j} = \rho_{ij}\sigma_{y_i}\sigma_{y_j}$.

Das χ^2 wird dann gemäss der allgemeinen Formel

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

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

Korrelationen zwischen Parametern

Im allgemeinen sind die resultierenden Fit Parameter $\vec{a}$ korreliert.

Häufig stellt sich das Problem daraus eine weitere Grösse $b = f(\vec{a})$ zu bestimmen und ihren Fehlerσ_b.

Für 2 Parameter a_0, a_1 ergibt sich

$$\sigma_b^2 = \sigma_{a_0}^2 \left(\frac{\partial f}{\partial a_0}\right)^2 + \sigma_{a_1}^2 \left(\frac{\partial f}{\partial a_1}\right)^2 + 2\, cov(a_0, a_1) \frac{\partial f}{\partial a_0} \frac{\partial f}{\partial a_1}$$

Oder im allgemeinen Fall für m Parameter:

$$\sigma_b^2 = \sum_i^m \sum_j^m \frac{\partial f}{\partial a_i} \frac{\partial f}{\partial a_j} V_{ij}$$

mit der Fehler-Matrix V_{ij}.

Beispiel

Beispielsweise erhält man aus einem Geradenfit $y = a_0 + a_1 \cdot x$ den Achsenabschnitt a_0 und die Steigung a_1.

Gesucht sei $b = y(x_0) = a_0 + a_1 \cdot x_0$.

Dann ist

$$\sigma_b^2 = \sigma_{a_0}^2 + \sigma_{a_1}^2 \, x_0^2 + 2 \, cov(a_0, a_1) \, x_0$$

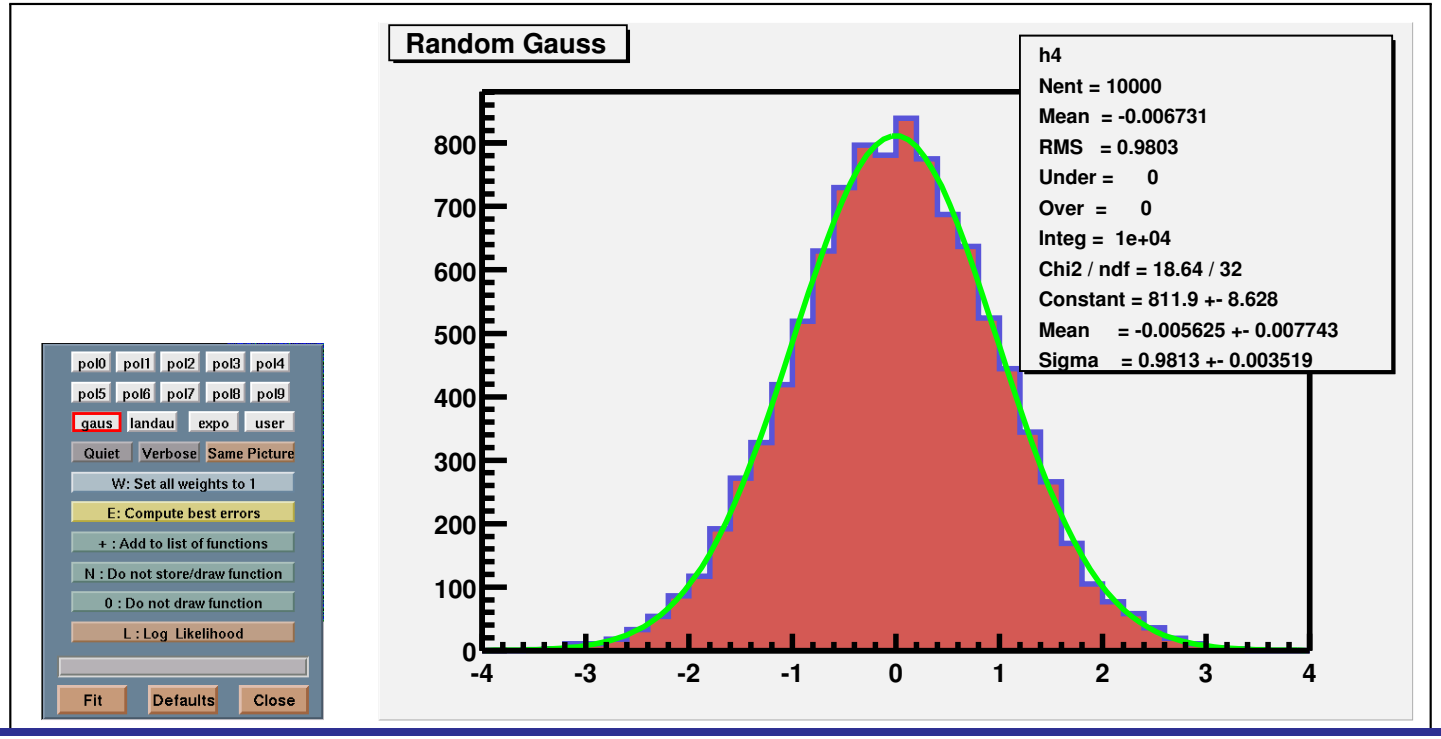
Fitten mit ROOT

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 ...).

Fitten mit ROOT

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

$\rightarrow$ 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.



Fitten mit ROOT

Per default wird ein χ^2 fit durchgeführt; für Histogramme wird dabei als Fehler für ein Bin mit N Einträgen $\sqrt{N}$ benutzt. Alternativ kann man einen 'binned maximum likelihood' Fit durchführen (Log Likelihood), dann wird die Poissonstatistik für die Bin-Einträge verwendet.

Fitten mit vordefinierten Funktionen

Einige Funktionen sind vordefiniert in ROOT:

- Polynome bis zum 9. Grad
- Gauss
- Exponentialfunktion
- Landau Verteilung

Fitten mit Benutzer definierten Funktionen

Für viele Fälle kann man auf die in ROOT eingebauten Funktionen zurückgreifen, allerdings müssen dann brauchbare Startwerte für den Fit vorgegeben werden.

```
Root > TF1 *myfit = new TF1("myfit","[0]*sin(x) + [1]*exp(-[2]*x)",0,2);
// set parameters
Root > myfit->SetParameters(1,0.05,0.2);
// Fitten
Root > hist->Fit("myfit");
```

Fitten mit Benutzer definierten Funktionen

In komplizierteren Fällen kann man auch eigene Funktionen definieren:

```
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 > TF1 *bw = new TF1("bw",BreitWig, 70, 90, 2);
// set parameters
Root > bw->SetParameters(80,2);
Root > hist->Fit("bw");
```

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
TMinuit *gMinuit = new TMinuit(npar);
// Tell Minuit the function
gMinuit->SetFCN(fcn);
// Start Werte fuer Minuit

gMinuit->mnparm(0, "a", 1.3, 0.1, 0.1, 5., ierflg);
// minimization
gMinuit->mnexcm("MIGRAD", arglist ,2,ierflg);
```

Fitten mit MINUIT

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 ist zunächst nur der pointer/array `par`, in dem die aktuellen Werte der Parameter übergeben werden. Und das Argument `f`, mit dem das berechnete χ^2 oder $\mathcal{L}$ zurückgegeben wird.

Die Funktion `fcn` wird intern von MINUIT gerufen. In `fcn` braucht man Zugang zu den Messwerten und Fehlern. Am einfachsten geht das über **global** deklarierte pointer, arrays oder Variablen.

Zum Erstellen und Invertieren der Kovarianzmatrix $\mathcal{C}$ empfiehlt sich die ROOT Klasse `TMatrix`.

Beispiel

```
// fcn function for fit with covariance matrix
double arr_poi;    // global array pointer
TMatrix *icovp;    // global matrix pointer
int nval = 3;      // global for number of measurements
int main()
{
  arr_poi = new double[3];
  TMatrix cov(3,3);
```

Beispiel

```
  cov(0,0) = ...    , cov(2,2) = ...;
  // Invert matrix
  TMatrix icov = cov.Invert();
  icovp = &icov;
  ...;
}
void fcn(Int_t &npar,Double_t *gin,Double_t &f,
           Double_t *par,Int_t iflag)
{
  ...
  for (int i; i < nval; i++ ) {
    for (int j; j < nval; j++ ) {
      chi += (arr_poi[i] - fit_func(i)) *
             (arr_poi[j] - fit_func(j)) * (*icovp)(i,j) ;

    }
  }
...
}
```