Script: Anpassen von Funktionen an Messdaten

23. März 2015

Funktionsanpassung mit der χ^2-Methode

Zusammenfassung

Der Vergleich von Modellen mit Messungen gehört zu den Standardaufgaben in der Experimentalphysik. Im einfachsten Fall stellt ein Modell eine Funktion dar, die Vorhersagen für Messdaten liefert und die üblicherweise von Modellparametern abhängt. Neben der Überprüfung, ob das Modell die Daten beschreibt, gehört die Bestimmung der Modellparameter zu den typischen Aufgaben. Zur Funktionsanpassung wird häufig die Methode der kleinsten Quadrate verwendet, mit der sich sehr elegant auch korrelierte Unsicherheiten oder Fehler in Abszissenrichtung zusätzlich zu Fehlern in Ordinatenrichtung behandeln lassen. Dieses Script gibt einen kurzen Abriss der Methode und enthält praktische Hinweise zur Funktionsanpassung mit Hilfe numerischer Methoden auf dem Computer.

Prof. Dr. Günter Quast
Homepage: `http://www.ekp.kit.edu/~quast`
E-Mail: `G.Quast@kit.edu`

Inhaltsverzeichnis

1 Anpassung von Funktionen an Datenpunkte **2**

2 χ^2-Methode zur Funktionsanpassung **3**

2.1 Behandlung von korrelierten Fehlern 4

2.2 Bestimmung der Unsicherheiten der Parameter 5

2.3 χ^2 als Testgröße für die Qualität einer Anpassung 6

3 Konstruktion von Kovarianzmatrizen **6**

3.1 Korrelationsmatrizen 7

4 Analytische Lösung für lineare Probleme **8**

4.1 Lineare Regression 8

5 Grenzen der χ^2-Methode **9**

6 Likelihood und χ^2-Methode **10**

7 Bestimmung der Parameterunsicherheiten im nicht-linearen Fall **10**

8 Abschließende Anmerkungen **12**

A Programme zur Funktionsanpassung **13**

A.1 Funktionsanpassung mit *qtiplot* 14

A.2 Funktionsanpassung mit *gnuplot* 15

A.3 Funktionsanpassung mit *ROOT* 16

A.3.1 Funktionsanpassung in Root: C++ – Makro 16

A.3.2 Funktionsanpassung in Root: Python–Makro 17

A.4 Funktionsanpassung mit dem Python-Paket *kafe* 19

A.5 Funktionsanpassung mit *RooFiLab* 21

A.5.1 Installation 22

A.5.2 Bedienung des Programms *RooFiLab* 22

1 Anpassung von Funktionen an Datenpunkte

Zum Vergleich von Messdaten mit theoretischen Modellen oder zur Bestimmung von Parametern werden Funktionen an Messdaten angepasst. Zu den gegebenen N Messwerten (x_1, y_i), $i = 1, \ldots, N$ wird also eine Funktion f gesucht, deren Funktionswerte $f(x_i)$ an den Stützstellen x_i möglichst nahe bei den Werten y_i liegt. Oft wird eine bestimmte Form der Funktion vorgegeben, z. B. ein Polynom, eine Exponentialfunktion o. Ä., die durch die theoretische Erwartung vorgegeben ist, und die eine Anzahl von K freien Parametern p_j enthält mit $j = 1, \ldots, K; K < N$. Abbildung 1 zeigt das Beispiel einer Parabel, die an mit Fehlern in Richtung der Ordinaten-Achse behaftete Messpunkte angepasst wurde.

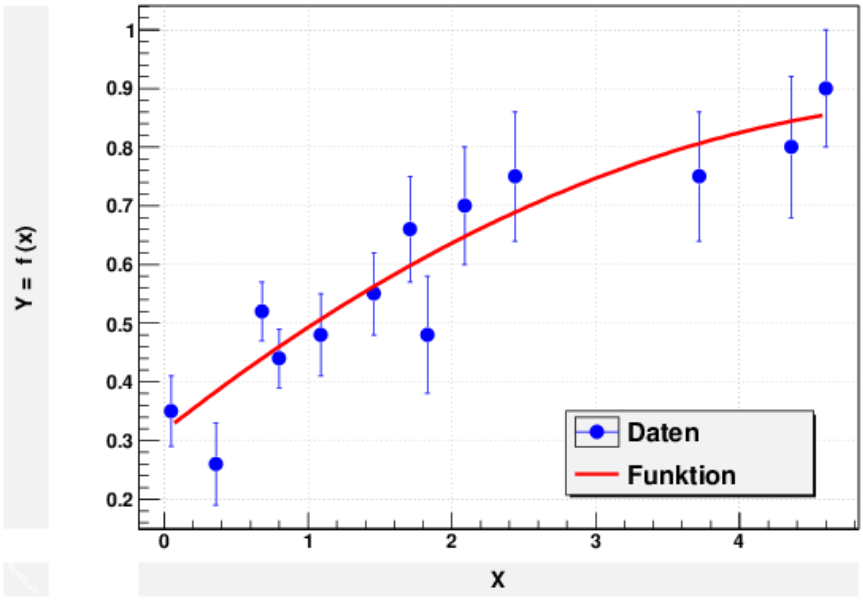

Abbildung 1: Beispiel der Anpassung einer Funktion (hier $f(x) = ax^2 + bx + c$) an Messpunkte (x_i, y_i).

Zum Auffinden der besten Werte der Parameter wird ein geeignetes Abstandsmaß benötigt. Fasst man die auf den Stützstellen x_i definierten Funkionen als Elemente eines Vektorraums auf, also $\vec{y} = (y_1, \ldots, y_N)$ und $\vec{f} = (f(x_1; p_1, \ldots, p_K), \ldots, f(x_1; p_1, \ldots, p_K))$, so bietet das Skalarprodukt dieser Vektoren ein solches Abstandsmaß, dass von den Elementen p_j des Parametervektors $\boldsymbol{p}$ abhängt: $d(\boldsymbol{p})^2 = (\vec{y} - \vec{f}(\boldsymbol{p})) \cdot (\vec{y} - \vec{f}(\boldsymbol{p}))$.

Für den so definierten Abstand von zwei Funktionen besteht die Aufgabe also darin, die besten Werte der Elemente des Parametervektors $\boldsymbol{p}$ zu finden. Dies gelingt in einfachen Fällen analytisch, d. h. durch die Bestimmung der Nullstellen der ersten Ableitungen von $d(\boldsymbol{p})^2$ nach den Parametern p_j. Meist wird die Minimierung jedoch mit Hilfe von numerischen Optimierungsmethoden vorgenommen, wie sie in gängigen Programmen wie *gnuplot* (http://www.gnuplot.info/), *Origin* (nur für Windows, aktuelle Version lizenzpflichtig), *qtiplot* (http://wiki.ubuntuusers.de/qtiplot, unter Linux frei verfügbar, Bedienkonzept dem von *Origin* nachempfunden) und das Datenanalyse-Framework *Root* (im Quellcode verfügbar, http://root.cern.ch) sowie darauf aufbauenden Anwendungen bzw. Programmpaketen implementiert sind. Auf einige dieser Programme wird in Abschnitt A etwas näher eingegangen.

Für statistische Daten, wie sie auch Messungen mit Messfehlern darstellen, muss das Abstandsmaß natürlich die Messfehler berücksichtigen: Messpunkte mit großen Fehlern dürfen weiter von der anzupassenden Funktion entfernt sein als solche mit kleinen Unsicherheiten.

2 χ^2-Methode zur Funktionsanpassung

Häufig wird zur Funktionsanpassung an Datenpunkte mit Fehlern die „Methode der kleinsten Quadrate“ verwendet, d. h. die Minimierung der Summe der quadratischen Abweichungen der Datenpunkte von der Fit-Funktion normiert auf die jeweiligen Messfehler. Als Abstandsmaß verwendet man in diesem Fall ein Skalarprodukt mit Gewichten, die dem Inversen der quadrierten Messfehler σ_i der Messungen entsprechen:

$$S = \left(\vec{y} - \vec{f}(\boldsymbol{p})\right) \cdot \left(\vec{y} - \vec{f}(\boldsymbol{p})\right) = \sum_{i=1}^{N} \left(y_i - f(x_i;\boldsymbol{p})\right)\, w_i\, \left(y_i - f(x_i;\boldsymbol{p})\right)\ ,\ \text{mit}\ w_i = \frac{1}{{\sigma_i}^2}\,.$$

Man nennt S auch die „gewichtete Summe von Residuenquadraten“(engl. „weighted sum of squared rediduals, WSSR“). Für Gauß-förmig um den wahren Wert verteilte Messfehler folgt S einer χ^2-Verteilung mit einer Anzahl von Freiheitsgraden, die durch die Zahl der Messwerte reduziert um die Zahl der anzupassenden Parameter, also $n_f = N - K$, gegeben ist, $\mathrm{pdf}(S) = \chi^2(S; N-K)$. Daher hat sich für diese Methode auch der Name „χ^2-Methode“ etabliert.

Geschrieben mit den Messfehlern σ_i und etwas umgeformt ergibt sich

$$\chi^2(\boldsymbol{p}) = \sum_{i=1}^{N} \left(\frac{y_i - f(x_i;\boldsymbol{p})}{\sigma_i}\right)^2\ . \tag{1}$$

Anmerkung 1: Falls $f(x_i, \boldsymbol{p})$ den Erwartungswert einer Messung am Punkt x_i beschreibt und die Messwerte einer Gaußverteilung mit Breite σ_i um diesen Erwartungswert folgen, so handelt es sich bei den einzelnen Summanden um die Quadrate von sogenannten „reduzierten Zufallsvariablen“, die einer Standard-Normalverteilung folgen, d. h. einer Gaußverteilung mit Mittelwert Null und einer Standardabweichung von Eins. Der Zusammenhang mit der χ^2-Verteilung wird dadurch unmittelbar klar: die χ^2-Verteilung mit n_f Freiheitsgraden beschreibt ja gerade die Verteilung der Summe der Quadrate von n_f standardnormalverteilten Zufallszahlen.

Anmerkung 2: Für um den Erwartungswert $f(x_i)$ Gauß-förmig verteilte Messwerte y_i ist die χ^2-Methode äquivalent zum Likelihood-Verfahren, das die vollständige Kenntnis der Verteilungsdichte der Messfehler voraussetzt. Die χ^2-Methode benötigt dagegen nur den Erwartungswert und die Standardabweichung der Messfehlerverteilung und ist daher allgemeiner anwendbar.

Es lässt sich zeigen, dass die χ^2-Methode eine optimale und unverzerrte Schätzung der Parameter liefert, wenn die Parameter die Koeffizienten einer Linearkombination von Funktionen sind (siehe Abschnitt 4 zur analytischen Lösung solcher Probleme). Als Voraussetzung muss lediglich sichergestellt sein, dass die Varianzen der Messfehlerverteilungen existieren.

Unsicherheiten der Datenpunkte bzgl. der Abszissenachse, d. h. Fehler der x_i, können mit der χ^2-Methode recht elegant durch Iteration berücksichtigt werden:

- zunächst erfolgt eine Anpassung ohne Abszissenfehler,
- im zweiten Schritt werden diese dann mit Hilfe der ersten Ableitungen $f'(x_i)$ der im ersten Schritt angepassten Funktion f in entsprechende Fehler der Ordinate umgerechnet und quadratisch zu den betreffenden Fehlern der Ordinate addiert: ${\sigma_i}^2 = {\sigma_{y_i}}^2 + (f'(x_i) \cdot \sigma_{xi})^2$. Die Größe χ^2 wird jetzt mit den neuen Fehlern σ_i berechnet und minimiert.
- Ein dritter Schritt, der der Vorgehensweise beim zweiten Schritt entspricht, dient zur Verbesserung des Ergebnisses und zur Fehlerkontrolle - der Wert von χ^2 am Minimum darf sich vom zweiten zum dritten Schritt nicht signifikant ändern, ansonsten muss nochmals iteriert werden.

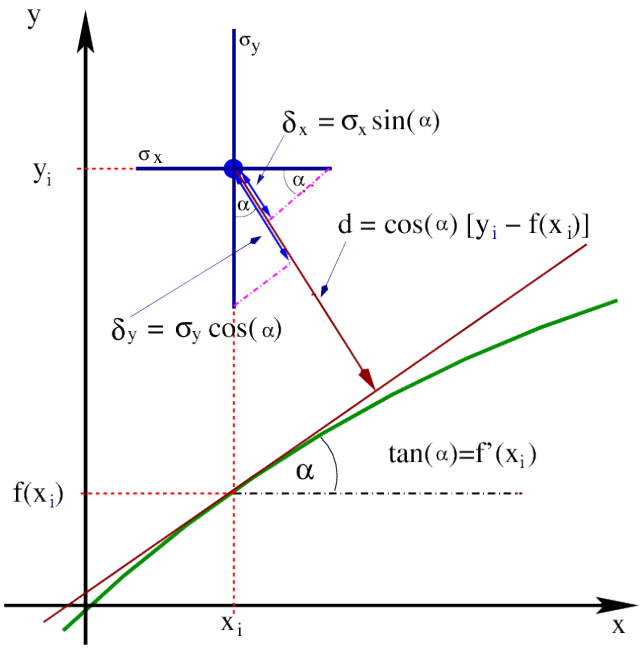

Abbildung 2: Illustration des Abstandsmaßes für einen Punkt mit Fehlern in Richtung der Ordinaten- und Abszissenachse. Das im Text beschriebene Verfahren entspricht der Minimierung des normierten Abstands der Messpunkte $(x_i,\, y_i)$ von der Tangente durch den Punkt $(x_i,\, f(x_i))$

Eine geometrische Interpretation dieser Vorgehensweise zeigt Abbildung 2. Die anzupassende Funktion wird durch die Tangente im Punkt (x_i,y_i) angenähert und das Quadrat des auf den Messfehler normierte Abstands der Funktion von den Datenpunkten, $\chi^2 = d^2 / \left(\cos^2(\alpha)\, {\sigma_y}^2 + \sin^2(\alpha)\, {\sigma_x}^2\right)$ minimiert. Mit $d = \cos(\alpha)\,(y_i - f(x_i))$ und $\tan\alpha = f'(x_i)$ folgt die oben angegebene Formel. Wenn die Tangente keine gute Näherung der anzupassenden Funkion über den Bereich der Messfehler von x_i ist, wird dieses Verfahren ungenau.

2.1 Behandlung von korrelierten Fehlern

Unsicherheiten von Messwerten sind typischerweise einerseits bestimmt durch unabhängige Fehler jeder Einzelmessung, wie z.B. Ablesefehler, statistische Fehler usw.. Andererseits gibt es systematische Fehler, die alle Messwerte in gleicher Weise betreffen, also zwischen diesen „korreliert“ sind. Die Beschreibung solcher korrelierter Fehler geschieht mit Hilfe der sogenannten Kovarianz-Matrix C, einer symmetrischen Matrix, deren Dimension der Anzahl der Messungen entspricht. Die Diagonalelemente enthalten die Varianzen, d. h. den quadrierten Gesamtfehler der Messwerte, $C_{ii} = {\sigma_i^t}^2$, die Nebendiagonalelemente enthalten die gemeinsamen Fehlerkomponenten, $C_{ij} = \sigma_i^g \cdot \sigma_j^g$ der Messungen mit den Indizes i und j.

Wenn die Messfehler korreliert sind, wird χ^2 mit Hilfe der Kovarianzmatrix der Messfehler $\mathbf{C}$ ausgedrückt. Fasst man die Differenzen $y_i - f(x_i;\boldsymbol{p})$ zum sogenannten Residuenvektor mit den Komponenten zusammen, $\Delta_i(\boldsymbol{p}) = y_i - \mathrm{f}(x_i;\boldsymbol{p})$, so ergibt sich

$$\chi^2(\boldsymbol{p}) = \vec{\Delta}(\boldsymbol{p})^T\, \mathbf{C}^{-1}\, \vec{\Delta}(\boldsymbol{p})\,.$$

Hierbei ist C^{-1} die Inverse der Kovarianzmatrix.

Korrelierte Unsicherheiten der Messpunkte in Richtung der Abszisse werden durch eine Kovarianzmatrix mit den Elementen C_{ij}^x beschrieben und mit Hilfe der 1. Ableitung zu den Kovarianzmatrixelementen C_{ij}^y der

Ordinatenfehler addiert, d. h. die Elemente der gesamten Kovarianzmatrix $\mathbf{C}$ ergeben sich zu

$$C_{ij} \;=\; C^y_{ij} \;+\; C^x_{ij} \cdot f'(x_i) \cdot f'(x_j)\,.$$

Mit dieser neuen Kovarianzmatrix wird nun die Anpassung wiederholt und ein dritter Schritt zur Kontrolle der numerischen Fehler durchgeführt.

Insgesamt ergibt sich also mit dem Vektor der ersten Ableitungen $\vec{f'}$ der allgemeinste Ausdruck zur Berücksichtigung von korrelierten Unsicherheiten der Messpunkte in Abszissen- und Ordinatenrichtung mit Kovarianzmatrizen $\mathbf{C}^x$ bzw. $\mathbf{C}^y$:

$$\chi^2(\boldsymbol{p}) = \vec{\Delta}(\boldsymbol{p})^T \left(\mathbf{C}^y \;+\; \mathbf{C}^{x'})^{-1}\right) \vec{\Delta}(\boldsymbol{p}) \;\text{ mit }\; (\mathbf{C}^{x'})_{i,j} = C^x_{i,j} f'(x_i) f'(x_j)\,. \tag{2}$$

Dieses allgemeine Verfahren ist im Programm *RooFiLab* implementiert.

2.2 Bestimmung der Unsicherheiten der Parameter

Anschaulich hängen die Unsicherheiten der Parameter davon ab, wie scharf das Minimum um den Wert der besten Anpassung ist, d. h. je größer die Krümmung am Minimum, desto kleiner die Unsicherheiten. Daher hängen die Unsicherheiten der Parameter mit den zweiten Ableitungen von $\chi^2(\boldsymbol{p})$ nach den Parametern p_j zusammen, die bei den Werten $\hat{p}_i$ der Parameter am Minimum, dem best-fit-Punkt, ausgewertet werden. Die Krümmungen am Minimum legen die Elemente der Inversen der Kovarianzmatrix V_{ij} der Parameter fest:

$$V_{ij}{}^{-1} = \frac{1}{2}\,\frac{\partial^2 \chi(\boldsymbol{p})^2}{\partial p_i\,\partial dp_j}\bigg|_{\hat{p}_i \hat{p}_j}\,. \tag{3}$$

Werden mehrere Parameter angepasst, so sind deren Fehler häufig korreliert, selbst wenn die Fehler der Messdaten unkorreliert sind; dies ist oft unerwünscht, weil Ergebnisse nur unter Angabe aller mit ihnen korrelierten Parameter verwendbar sind. Durch geeignete Parametrisierung kann aber oft die Korrelation verringert werden. Zum Beispiel ist es bei der Anpassung von Geraden an Messpunkte viel sinnvoller, statt der gewohnten Darstellung $f(x) = ax + b$ eine Parametrisierung in den transformierten Variablen $y - \bar{y}$ und $x - \bar{x}$ vorzunehmen, wobei $\bar{x}$ und $\bar{y}$ die Mittelwerte der x- bzw. y-Werte bedeuten. Es ergibt sich dann als Geradengleichung $f(x) = a\,(x - \bar{x}) + b'$

Werden keine Fehler der Datenpunkte angegeben, so werden in der χ^2-Summe alle Gewichte, d. h. die Fehler σ_i der Messwerte, auf Eins gesetzt und eine Anpassung mit gleichem Gewicht aller Messpunkte durchgeführt. In diesem Fall sind die von manchen Programmen ausgegebenen Fehler der Parameter mit einiger Vorsicht zu betrachten, da sie unter sehr speziellen Annahmen bestimmt werden. Bei unbekannten oder nicht spezifizierten Unsicherheiten der Datenpunkte kann eine Abschätzung der Fehler der Parameter über den Wert von χ^2 am Minimum gewonnen werden, dessen Erwartungswert $\left\langle \chi^2(\hat{\boldsymbol{p}}) \right\rangle = N - K$ ist. Anders ausgedrückt, $\chi^2/n_f = \frac{\left\langle \chi^2(\hat{\boldsymbol{p}}) \right\rangle}{N-K}$ hat den Wert Eins. Werden für die Messpunkte in etwa gleiche Fehler vermutet, so führt man zunächst eine Anpassung mit $\sigma_i = 1$ durch und erhält χ^2/n_f. Die von der Anpassung gelieferten Fehler werden nun so skaliert, dass $\chi^2/n_f \equiv 1$ gilt.

Bei diesem Verfahren werden die Fluktuationen der Messwerte um die angepasste Kurve benutzt, um Aussagen über die Unsicherheiten der Datenpunkte zu erhalten, die mitunter auf andere Art nur schwer zu bestimmen sind. Allerdings ist die dabei gemachte Annahme identischer Fehler aller Datenpunkte in der Regel nicht richtig. Außerdem verliert man dadurch die Möglichkeit, aus dem Wert von χ^2 am Minimum eine Aussage über die Übereinstimmung der Messdaten mit dem gewählten Modell zu gewinnen, wie im folgenden Kapitel beschrieben wird. Bei einigen Programmpaketen, die nicht primär von Physikern genutzt werden, ist dieses Vorgehen allerdings als Standard voreingestellt.

2.3 χ^2 als Testgröße für die Qualität einer Anpassung

Da bei Gauß-förmiger Fehlerverteilung der Datenpunkte die Werte von χ^2 am Minimum einer χ^2-Verteilung folgen (s. Abbildung 3), kann dieser Wert als Test für die Güte der Beschreibung der Daten durch die Funktion benutzt werden. Dazu integriert man die χ^2-Verteilung vom beobachteten Wert bis ∞, und erhält so eine Aussage darüber, mit welcher Wahrscheinlichkeit ein schlechterer Wert χ_{min} von χ^2 am Minimum erwartet würde als tatsächlich beobachtet. Dies wird oft als „χ^2-Wahrscheinlichkeit“ bezeichnet:

$$\chi^2{}_{\rm prob} = \int_{\chi_{min}}^{\infty} \chi^2(s;n_f)\,{\rm d}s \;=\; 1. - \int_0^{\chi_{min}} \chi^2(s;n_f)\,{\rm d}s \quad {}^{1}. \tag{4}$$

In *ROOT* zum Beispiel ist diese Testgröße über die Funktion `Double_t Prob(Double_t chi2, Int_t ndf)` verfügbar.

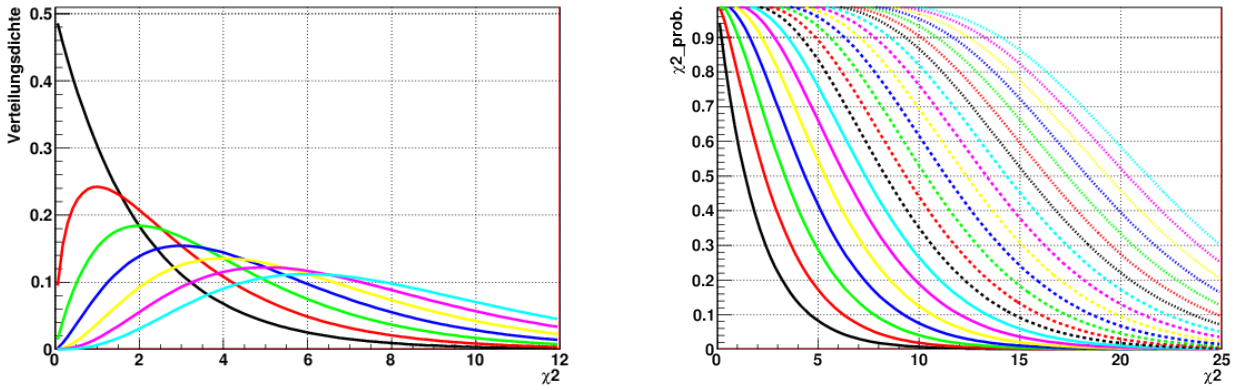

Abbildung 3: χ^2-Verteilung (links) für 2 – 8 Freiheitsgrade und χ^2-Wahrscheinlichkeit für 2 – 22 Freiheitsgrade (rechts). Der Erwartungswert der χ^2-Verteilung ist n_f und ihre Standardabweichung ist $\sqrt{2n_f}$. Für eine sehr große Anzahl von Freiheitsgraden geht die Verteilung in eine Gauß-Verteilung über. Durch Integration der Verteilungsdichte erhält man die im Text definierte χ^2-Wahrscheinlichkeit als Testgröße für die Güte einer Anpassung.

Für ein korrektes Modell ist sie im Intervall $[0, 1]$ gleichverteilt, d. h. z. B. dass in 5 % der Fälle auch bei korrektem Modell eine χ^2-Wahrscheinlichkeit von 0.05 oder kleiner beobachtet wird.

Anschaulich leichter zu handhaben ist der auf die Zahl der Freiheitsgrade normierte Wert χ^2/n_f mit einem Erwartungswert der Verteilung von Eins, wie in Abbildung 4 gezeigt. Mit wachsender Zahl der Freiheitsgrade wird die Streuung der Verteilung um den Wert 1. kleiner, die Breite ist $\frac{2}{\sqrt{n_f}}$. Bei 20 Freiheitsgraden wir nur mit einer Wahrscheinlichkeit von 10 % ein Wert von χ^2/n_f größer als 1.5 erwartet.

3 Konstruktion von Kovarianzmatrizen

Die Kovarianzmatrix C ist eine quadratische und symmetrische Matrix, deren Dimension die Anzahl der Messwerte N hat. Die Diagonalelemente der Kovarianzmatrix sind durch den Gesamtfehler der Messwerte y_i gegeben:

$$C_{ii} = {\sigma_i^{(t)}}^2$$

[1] Der 2. Ausdruck ist numerisch einfacher zu berechnen.

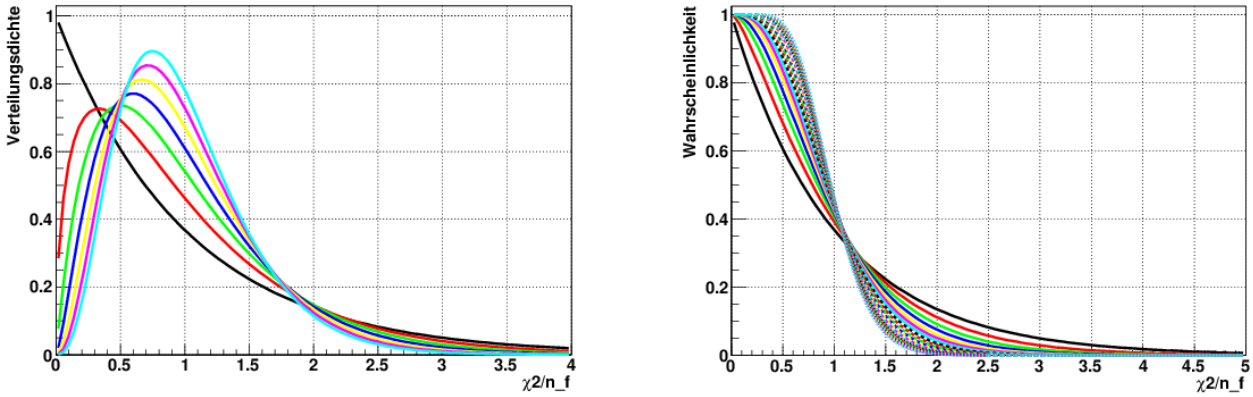

Abbildung 4: Verteilungsdichte von χ^2/n_f (links) für 2 – 8 Freiheitsgrade und die entsprechende Wahrscheinlichkeit für 2 – 22 Freiheitsgrade, einen größeren Wert als den auf der x-Achse angegebenen zu finden (rechts).

Die Produkte der korrelierten Fehlerkomponenten $\sigma_i^{(g)}$ und $\sigma_j^{(g)}$ der Messwerte y_i und y_j bilden die Nebendiagonalelemente:

$$C_{ij} = \sigma_i^{(g)}\,\sigma_j^{(g)}$$

Sind zum Beispiel alle Messwerte von einem gemeinsamen Fehler $\sigma^{(g)}$ betroffen, so gilt $C_{ij} = {\sigma^{(g)}}^2$ für alle i, j. Es können auch Gruppen von Messungen von gemeinsamen Fehlern $\sigma^{(g_K)}$ betroffen sein; dann stehen die Quadrate dieser Fehler jeweils in den zu den Blockmatrizen der Gruppe k gehörenden Nebendiagonalelementen. Im allgemeinsten Fall müssen die korrelierten Fehleranteile $\sigma_i^{(g)}$ und $\sigma_j^{(g)}$ nicht gleich sein. Das ist zum Beispiel dann der Fall, wenn die Messfehler durch einen relativen Anteil der Messwerte gegeben sind, also beispielsweise ein korrelierter Fehler von 1 % des jeweiligen Messwertes vorliegt.

Bei der Konstruktion der Kovarianzmatrix beginnt man mit den unkorrelierten Fehlern ${\sigma^{(u)}}_i$ der Messwerte und setzt deren Quadrate auf die Diagonale. Solche unkorrelierten Fehler sind häufig die statistischen Fehler einer Messung. Die korrelierten Fehler ${\sigma^g}_i$ und ${\sigma^g}_j$, häufig von systematischen Fehlern herrührend, werden quadratisch zum jeweiligen Diagonalelement addiert und auch auf der Nebendiagonalen eingetragen:

$$\begin{aligned} C_{ii} &= {\sigma_i^{(t)}}^2 = {\sigma_i^{(u)}}^2 + {\sigma_i^{(g)}}^2 \\ C_{jj} &= {\sigma_j^{(t)}}^2 = {\sigma_j^{(u)}}^2 + {\sigma_j^{(g)}}^2 \\ C_{ij} &= \sigma_i^{(g)}\sigma_j^{(g)} \\ C_{ji} &= C_{ij} \end{aligned}$$

Wenn es mehrere korrelierte Fehlerkomponenten gibt, so erhält man die Gesamtfehler-Kovarianzmatrix durch Addition aller so berechneten Kovarianzmatrizen. Dies entspricht der quadratischen Addition von Einzelfehlerbeiträgen – Kovarianzmatrixelemente sind quadratische Formen!

3.1 Korrelationsmatrizen

Häufig verwendet man statt der Kovarianzmatix die sogenannte Korrelationsmatrix Cor mit den Elementen

$$Cor_{ij} = \frac{C_{ij}}{\sqrt{C_{ii}\,C_{jj}}} = \frac{C_{ij}}{\sigma_i\,\sigma_j}\,.$$

Alle Diagonalelemente der Korrelationsmatrix sind 1, und für die Nebendiagonalelemente gilt $-1 < Cor_{ij} < 1$. Ist Cor_{ij} Eins, so sind die Messungen y_i und y_j voll korreliert, für $Cor_{ij} = -1$, spricht man von vollständiger Antikorrelation. Wegen des eingeschränkten Wertebereichs der Matrixelemente sind Korrelationsmatrizen anschaulicher und leichter zu bewerten als Kovarianzmatrizen. Bei Kenntnis der Korrelationsmatrix müssen auch die Gesamtfehler der Messwerte bekannt sein, um die Kovarianzmatrix z. B. für die Verwendung in Parameteranpassungen zu konstruieren:

$$C_{ij} = \underbrace{\sigma_i \cdot \sigma_j}_{\sqrt{C_{ii} \cdot C_{jj}}} \cdot Cor_{ij}$$

4 Analytische Lösung für lineare Probleme

Wenn die Parameter nur linear in der gewichteten Summe S der Residuenquadrate auftreten, lässt sich das Minimum bzgl. des Parametervektors $\boldsymbol{p}$ analytisch bestimmen. Schreibt man die anzupassende Funktion f als Linearkombination von K Funktionen F_j mit $f(x_i) = \sum_{j=1}^{K} p_j F_j(x_i)$ und führt die $N \times K$-Matrix A mit N Zeilen und K Spalten mit den Koeffizienten $A_{ij} := F_j(x_i)$ ein, so vereinfacht sich der Residuenvektor zu $\vec{\Delta}(\boldsymbol{p}) = \vec{y} - A\boldsymbol{p}$. Für S ergibt sich also mit $W = C^{-1}$:

$$S(\boldsymbol{p}) = (\vec{y} - A\boldsymbol{p})^T W (\vec{y} - A\boldsymbol{p}). \tag{5}$$

Das Minimum findet man durch Nullstellenbestimmung der ersten Ableitung, $\left.\frac{\mathrm{d}S}{\mathrm{d}\boldsymbol{p}}\right|_{\hat{\boldsymbol{p}}} = 0$, und Auflösen nach dem gesuchten Parametervektor $\hat{\boldsymbol{p}}$. Die Lösung ist

$$\hat{\boldsymbol{p}} = (A^T W A)^{-1} A^T W \vec{y}. \tag{6}$$

Die Schätzwerte für die Parameter ergeben sich also durch Linearkombination der Messwerte mit Koeffizienten, in die die Kovarianzmatrixelemente der Messungen und die Funktionswerte $F_j(x_i)$ eingehen.

Die Kovarianzmatrix der Parameter erhält man durch Fehlerfortpflanzung der Kovarianzmatrix C der Messfehler. Mit der Abkürzung $B := (A^T W A)^{-1} A^T W$ gilt $\hat{\boldsymbol{p}} = B\vec{y}$; damit ergibt sich die Kovarianzmatrix der Parameter zu $V_{\hat{\boldsymbol{p}}} = B^T C B$, also nach einigen Vereinfachungen

$$V_{\hat{\boldsymbol{p}}} = (A^T W A)^{-1} = (A^T C^{-1} A)^{-1}. \tag{7}$$

Alternativ hätte man natürlich, wie oben schon beschrieben, die mit $\frac{1}{2}$ multiplizierte Inverse der Matrix der zweiten Ableitungen von S nach den Parametern bilden können, mit identischem Ergebnis.

4.1 Lineare Regression

Aus dem hier erhaltenen allgemeinen Ergebnis lassen sich die bekannten Formeln für die lineare Regression bei unkorrelierten Messfehlern gewinnen. Für die Anpassung einer Geraden $f(x) = p_1 + p_2\, x$ gilt

$$A = \begin{pmatrix} 1 & x_1 \\ \ldots & \ldots \\ 1 & x_N \end{pmatrix}, \quad W = \begin{pmatrix} \frac{1}{\sigma_1^2} & 0 & \ldots & 0 & 0 \\ 0 & \ldots & \frac{1}{\sigma_i^2} & \ldots & 0 \\ 0 & 0 & \ldots & 0 & \frac{1}{\sigma_N^2} \end{pmatrix}.$$

Man erhält durch Einsetzen in Gleichungen 6 und 7 mit den Abkürzungen

$$S_1 = \sum_{i=1}^{N} \frac{1}{\sigma_i^2}, \qquad S_x = \sum_{i=1}^{N} \frac{x_i}{\sigma_i^2} = \overline{x}\, S_1, \qquad S_y = \sum_{i=1}^{N} \frac{y_i}{\sigma_i^2} = \overline{y}\, S_1,$$

$$S_{xx} = \sum_{i=1}^{N} \frac{x_i^2}{\sigma_i^2} = \overline{x^2}\, S_1, \qquad S_{xy} = \sum_{i=1}^{N} \frac{x_i\, y_i}{\sigma_i^2} = \overline{xy}\, S_1, \qquad D = S_1 S_{xx} - S_x^2$$

als Lösung

$$\begin{aligned} \hat{p}_1 &= \tfrac{S_{xx}S_y - S_x S_{xy}}{D}\,, \qquad {\sigma_{p_1}}^2 = \tfrac{S_{xx}}{D}\,, \\ \hat{p}_2 &= \tfrac{S_1 S_{xy} - S_x S_y}{D}\,, \qquad {\sigma_{p_2}}^2 = \tfrac{S_1}{D}\,, \qquad V_{12} = \tfrac{-S_x}{D}\,. \end{aligned}$$

Das Kovarianzmatrixelement V_{12} verschwindet, wenn $S_x = 0$ gilt, der Erwartungswert $\overline{x}$ der Abszissenwerte also Null ist. Dies kann man durch geeignete Parametrisierung der Geradengleichung erreichen, wenn man $x' = x - \overline{x}$ setzt, d. h. $f'(x) = p'_1 + p'_2\,(x - \overline{x})$. Jetzt erhält man die einfacheren, unkorrelierten Lösungen

$$\begin{aligned} \hat{p}'_1 &= \tfrac{S_y}{S_1} = \overline{y}\,, \qquad {\sigma_{p'_1}}^2 = \tfrac{1}{S_1}\,, \\ \hat{p}'_2 &= \tfrac{S_{x'y}}{S_{x'x'}} = \tfrac{\overline{x'y}}{\overline{x'^2}}\,, \qquad {\sigma_{p'_2}}^2 = \tfrac{1}{S_{x'x'}}\,. \end{aligned}$$

Für die Weiterverwendung sind unkorrelierte Ergebnisse von großem Vorteil, so dass man zur linearen Regression immer dieses letztgenannte Verfahren anwenden sollte.

Die hier abgeleiteten Formeln finden in zahlreichen Computerprogrammen und auch in Taschenrechnern Verwendung und sind Bestandteil mancher Praktikumsanleitung. Oft werden Fehler der Messwerte nicht berücksichtigt, d. h. $\sigma_i = 1$ für alle i und damit $S_1 = N$. Das hier beschriebene Verfahren mit Berücksichtigung von Messfehlern wird in der Literatur üblicherweise als "gewichtete lineare Regression" bezeichnet.

Für die Lösung von Problemen, die nicht-linear in den Parametern sind, werden numerische Methoden zur Minimierung der χ^2-Funktion eingesezt. Es gibt zahlreiche Varianten solcher Optimierungsalgorithmen zur Bestimmung des Minimums einer skalaren Funktion in einem K-dimensionalen Parameterraum. Konkrete Implementierungen finden sich in diversen Softwarepaketen, von denen einige in Abschnitt A beschrieben werden.

5 Grenzen der χ^2-Methode

Bei vielen typischen Problemen in der Physik folgen die Unsicherheiten nicht der Gaußverteilung. Dazu gehören z. B. Experimente, bei denen Zählraten oder Häufigkeitsverteilungen gemessen werden. Hier folgt die Verteilung der Unsicherheiten einer Poisson-Verteilung, d. h. die Wahrscheinlichkeit n Ereignisse zu beobachten, wenn μ erwartet wurden, ist gegeben durch $P(n;\mu) = \frac{\mu^n}{n!}\mathrm{e}^{-\mu}$. Für große n nähert sich diese Verteilung einer Gaußverteilung mit Mittelwert μ und Breite $\sqrt{\mu}$ an. Die Unsicherheit hängt in diesem Fall auch vom Messwert selbst ab, dessen wahren Wert man aber nicht kennt. Abhängigkeiten der angenommenen Messunsicherheit vom Messwert treten auch bei allen Arten von relativen Fehlern auf, oder bei der Anwendung von fehlerbehafteten Korrekturfaktoren auf die gemessenen Werte.

In solchen Fällen ist Vorsicht geboten, wenn man die χ^2-Methode einsetzen möchte. Für das Beispiel Poisson-verteilter Unsicherheiten ergibt sich

$$S(\vec{n};\boldsymbol{p}) = \sum_{i=1}^{N} \frac{\left(n_i - \mu_i(\boldsymbol{p})\right)^2}{\mu_i(\boldsymbol{p})}\,.$$

Der Einfachheit halber setzt man für die Fehlerquadrate im Nenner oft die aus der Beobachtung gewonnen Werte n_i ein; dann jedoch erhält man eine stark verzerrte Anpassung: eine Fluktuation zu kleineren Werten führt zu einem kleineren Fehler, und das Gewicht in der Anpassung wird größer. In der Konsequenz wird die Anpassung also in Richtung der zu kleineren Werten fluktuierten Messungen verzerrt. Wenn es für einzelne Messungen i zu einer Beobachtungen von null Ereignissen kommt, kann dieser Messpunkt überhaupt nicht verwendet werden und muss weggelassen werden – obwohl auch eine solche Beobachtung Information enthält! Dieses Problem kann man durch Iteration vermeiden:
– in einem ersten Schritt wird eine Anpassung mit den beobachteten Fehlern durchgeführt,
– im zweiten Schritt werden die Fehlerquadrate durch die im ersten Schritt gewonnenen Werte aus der Anpassung, $\mu_i(\boldsymbol{p})$ ersetzt.

In vielen Fällen, also bei sehr kleinen Zählraten im obigen Beispiel, ist aber die korrekte Berücksichtigung des vollen Fehlermodells erforderlich. Dann sind die Grenzen der Anwendbarkeit der χ^2-Methode erreicht.

6 Likelihood und χ^2-Methode

Als Alternative bleibt sich die Anwendung der Likelihood-Methode an, die im folgenden am Beispiel einer Zählratenmessung kurz erläutert wird.

Zunächst berechnet man mit Hilfe der Poissonverteilung die Wahrscheinlichkeiten, in der Messung i den Wert n_i zu beobachten, und multipliziert die so erhaltenen Werte für alle Messungen. Man erhält dann die vom Parametervektor $\boldsymbol{p}$ der Dimension K abhängige Likelihood-Funktion $\mathcal{L} = \prod_{i=1}^{N} \mathrm{P}\left(n_i; \mu_i(\boldsymbol{p})\right)$. Gemäß dem Likelihood-Prinzip liefert die Maximierung der Likelihood-Funktion bzgl. der Parameter $\boldsymbol{p}$ eine Schätzung für die gesuchten Parameter.

In der Praxis verwendet man den negativen Logarithmus der Likelihood, lässt konstante, d. h. nicht vom Parametervektor abhängige Terme weg und erhält die „negative Log-Likelihood Funktion“ für das Problem,

$$-\log \mathcal{L}(\vec{n}; \boldsymbol{p}) \;=\; \sum_{i=1}^{N} -n_i \cdot \ln(\mu_i(\boldsymbol{p})) \;+\; \mu(\boldsymbol{p})\,, \tag{8}$$

die man bzgl. der Parameter minimiert.

Die Bestimmung der Parameterunsicherheiten kann wieder durch Analyse der zweiten Ableitungen am Minimum erfolgen:

$$V_{ij}{}^{-1} = \left.\frac{\partial^2 \ln \mathcal{L}(\vec{n}; \boldsymbol{p})}{\partial p_i \, \partial d p_j}\right|_{\hat{p}_i \hat{p}_j}\,. \tag{9}$$

Die Anwendung des Liklihood-Verfahrens immer immer dann notwendig, wenn die Verteilung der Unsicherheiten stark von der Gaußverteilung abweicht oder die von den Messwerten selbst abhängt. Für jeden Datenpunkt (x_i, y_i) muss dann die korrekte Wahrscheinlichkeitsdichte $\mathcal{P}_i(x_i, y_i; \boldsymbol{p})$ in Abhängigkeit von den Parametern $\boldsymbol{p}$ spezifiziert und durch Multiplikation der den beobachteten Messwerten entsprechenden Likelihood-Werte die Gesamt-Likelihood des Problems bestimmt werden:

$$-\log(\mathcal{L}) = -\sum_{i=1}^{N} \log\left(\, \mathcal{P}_i(x_i, y_i;\, \boldsymbol{p})\,\right) \tag{10}$$

Für um die wahren Werte Gauß-förmig verteilte Fehler entspricht $-\log \mathcal{L}$ übrigens bis auf einen Faktor $\frac{1}{2}$ der χ^2-Größe,

$$-\log \mathcal{L} = \frac{1}{2}\chi^2\,. \tag{11}$$

Die Maximierung der Likelihood, oder, völlig äquivalent, die Minimierung des negativen Logarithmus der Likelihood, erfordert in der Regel numerische Verfahren, da im Allgemeinen $-\log \mathcal{L}$ eine komplizierte Funktion der K Parameter ist.

7 Bestimmung der Parameterunsicherheiten im nicht-linearen Fall

Bei Anpassungsproblemen, bei denen die anzupassende Funktion nicht-linear von den Parametern abhängt, ist auch nicht garantiert, dass ein (K-dimensionales) Paraboloid eine gute Näherung der χ^2- bzw. Likelihood-Funktion am Minimum darstellt. Die korrekte Verallgemeinerung des Verfahrens zur Bestimmung der Parameterunsicherheiten besteht darin, den kompletten Verlauf der Likelihood-Funktion in der Nähe des Minimums zu

berücksichtigen. Das Fehlerintervall ergibt sich aus den Werten der Parameter, bei denen die Likelihood um den Wert $\Delta \log \mathcal{L} = \frac{1}{2}$ über dem Minimum liegt. Auch im allgemeinen, nicht-parabolischen Fall erhält man so ein Intervall mit 68 % Konfidenzniveau für die Werte der Parameter. Wenn nur einer der Parameter von Interesse ist, wird der Einfluss der anderen, evtl. mit diesem Parameter korrelierten Parameter dardurch berücksichtigt, dass man bei der Bestimmung von $\Delta \log \mathcal{L}$ bzgl. aller anderen Parameter (numerisch) minimiert, also die sogenannte „Profil-Likelihood“ verwendet. Im einfachen Fall eines parabolischen Verlaufs der Likelihood um das Minimum ist dieses Verfahren identisch zur Fehlerbestimmung mit Hilfe der zweiten Ableitungen am Minimum nach Formel 9. Wegen der Äquivalenz von χ^2- und Likelihood-Methode für Gauß-förmige Fehler der Eingabedaten nach Formel 11 gilt das Gesagte analog auch für die χ^2-Methode.

Bei ungünstig gewählter Parametrisierung können allerdings auch bei scheinbar „einfachen“ Fällen unerwartete Unterschiede zwischen den beiden Methoden zur Fehlerbestimmung auftreten, wie in Abbildung 5 am Beispiel der Anpassung einer Exponentialfunktion illustriert ist.

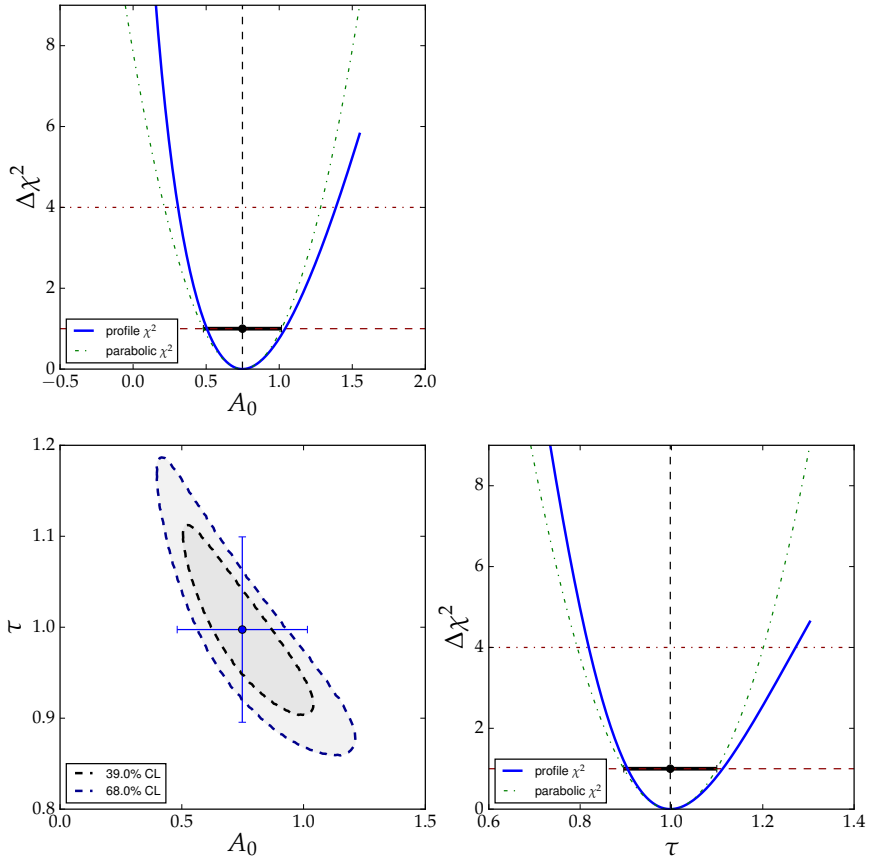

Abbildung 5: Beispiel der Anpassung einer Exponentialfunktion $A(t; A_0, \tau) = A_0 \exp\left(\frac{-t}{\tau}\right)$ (erzeugt mit dem unten beschriebenen Paket *kafe*, das auf dem am CERN entwickelten Minimierer *Minuit* beruht). Für die beiden Parameter der Funktion ist der Verlauf der Profil-Likelihood gezeigt, der mit den aus den Krümmungen am Minimum bestimmten Parabeln verglichen wird. Die im Bild links unten gezeigte, aus der Profil-Likelihood gewonnene Konfidenz-Kontur weicht ebenfalls sehr stark von der Ellipse ab, deren Achsen durch das aus den zweiten Ableitungen gewonnene Fehlerkreuz markiert sind. Übrigens: Wenn man statt des Parameters τ dessen Kehrwert $\gamma = \tau^{-1}$ verwendet, also die Parametrisierung $A(t; A_0, \gamma) = A_0 \exp(-t\gamma)$, so ist die Abweichung vom parabolischen Verhalten am Minimum deutlich kleiner.

Nur die wenigsten der gebräuchlichen Programmpakete unterstützen die Analyse der Profil-Likelihood, obwohl dies angesichts der heute verfügbaren Rechenleistung keine grundsätzliche Schwierigkeit mehr darstellt. Besonders bei Problemstellungen mit großen Unsicherheiten der Messgrößen empfiehlt es sich, die Gültigkeit der über die zweiten Ableitungen am Minimum gewonnenen Fehler zu überprüfen und ggf. durch die aus dem Scan der Profil-Likelihood gewonnen Fehler zu ersetzen.

8 Abschließende Anmerkungen

Die χ^2-Methode ist ein Spezialfall eines Liklelihood-Verfahrens für Gauß-verteilte Unsicherheiten der Eingangsgrößen. Wenn die Voraussetzung Gauß-förmiger Unsicherheiten in guter Näherung erfüllt ist, stellt die χ^2-Methode mit den oben in Abschnitt 4 abgeleiteten analytischen Lösungen für lineare Probleme und den unten in Abschnitt A vorgestellten numerischen Implementierungen ein elegantes und effizientes Verfahren zur Bestimmung der Parameter einer Modellfunktion dar.

Einige Anmerkungen zu häufig in der Praxis auftretende Schwierigkeiten und notwendige Erweiterungen der Methode seien hier kurz kommentiert:

- Da die χ^2-Methode Gauß-förmige Fehler der anzupassenden Datenpunkte voraussetzt, sollten die experimentellen Daten nie transformiert werden. Dank des zentralen Grenzwertsatzes sind die Unsicherheiten der Messdaten in der Regel gut durch eine Gaußverteilung beschrieben, und auch für Poisson-verteilte Ergebnisse von Zählexperimenten ist die Gauß'sche Näherung oft gut. Durch eine Transformation der Daten würde die Verteilungsdichte der Unsicherheiten ebenfalls verändert, und die Voraussetzungen für die Anwendung der χ^2-Methode sind unter Umständen nicht mehr gut erfüllt. Es sollte also immer die theoretische Modellierung so gewählt werden, dass das Modell die Messdaten beschreibt! Insbesondere Transformationen, die zu stark asymmetrischen Verteilungen der Unsicherheiten der Datenpunkte führen, müssen unbedingt vermieden werden!

- Häufig sind aus der theoretischen Modellierung stammende Korrekturen von Datenwerten notwendig, die zu von den Datenwerten abhängigen Unsicherheiten führen. Diese Unsicherheiten werden berücksichtigt, indem man sie quadratisch zu den Unsicherheiten der gemessenen Daten addiert. Dabei sollten allerdings zur Berechnung der Unsicherheiten nicht die gemessenen, sondern die aus der theoretischen Modellierung erwarteten Datenwerte eingesetzt werden, um eine Verzerrung in Richtung von statistisch zu Werten mit kleineren Unsicherheiten fluktuierenden Datenpunkten zu vermeiden.

- Die oben beschriebene χ^2-Wahrscheinlichkeit (Gleichung 4) ist nur bei Vorliegen von Gauß-verteilten Unsicherheiten ein guter Indikator für die Qualität einer Anpassung. Während die χ^2-Methode bzgl. des Zentralwerts und auch der Unsicherheiten der Parameter bei symmetrischen Verteilungsdichten der Datenunsicherheiten recht robust ist, reagiert der Wert von χ^2 am Minimum bzw. der Wert von χ^2/n_f oder der χ^2-Wahrscheinlichkeit sehr empfindlich auf solche Abweichungen. In der Praxis werden bei nicht Gauß'schen Fehlern daher oft sehr große Werte von χ^2/n_f noch als akzeptable Anpassung gewertet.

- Die χ^2-Methode wird auch als „Best Linear Unbiased Estimator (BLUE)“ bezeichnet. Das χ^2-Verfahren ist also für in den Parametern lineare Probleme das beste unter allen Verfahren zur Parameterschätzung, und außerdem ist es „unverzerrt“, was bedeutet, dass der Erwartungswert der Parameterschätzung (bestimmt aus einer großen Anzahl identischer Anpassungen auf statistisch unterschiedlichen Datenwerten) dem wahren Parameterwert entspricht.

- In vielen Fällen sind einige Parameter p_i eines Modells durch externe Bedingungen oder bereits erfolgte Messungen beschränkt, $p_i = p_i^0 \pm \sigma_{p_i}$. Das Einbauen solcher Randbedingungen an Parameter lässt sich durch Hinzufügen zusätzlicher Terme der Form

$$\frac{(p_i - p_i^0)^2}{\sigma_{p_i}}$$

zur χ^2-Funktion realisieren.

In der hier vorliegenden Darstellung wurden nur sehr einfache Beispiele behandelt, bei denen von wenigen Parametern abhängige Funktionen an Datenpunkte in einem zwei-dimensionalen Raum angepasst wurden. In der Praxis auftretende Probleme sind in der Regel sehr viel komplexer: Modelle werden an mehrere verschiedenartige Messungen angepasst, experimentelle Parameter, die für die theoretische Modellierung nicht interessant, für die Beschreibung des experimentellen Aufbaus aber wichtig sind, so genannte „Störparameter“, werden aus den Messdaten selbst oder aus Hilfsmessungen bestimmt und simultan angepasst [2]. Die Anpassung von einigen Hundert Parametern an Tausende von Messpunkten unterschiedlicher Art ist heute Standard. Fertige Anwendungen für solche komplexen Problemstellungen gibt es natürlich nicht, sondern die Parameter-abhängigen Likelihood-Funktionen werden von Physikern selbst programmiert und die Optimierung und statistische Interpretation unter Zuhilfenahme von Standardbibliotheken ausgeführt. Einige der unten aufgeführten, auf den Progamiersprachen C++ oder *Python* beruhende Programmpakete erlauben es, solche problemspezifischen Erweiterungen einzubauen.

A Programme zur Funktionsanpassung

Dank der Verbreitung von Computern können heute vollständige Minimierungsverfahren inklusive eines χ^2-Tests, einer Untersuchung der Korrelationen der angepassten Parameter und einer Analyse der Profil-Likelihood auch für komplexe nicht-lineare Probleme durchgeführt werden. Die Transformation von Messdaten zum Erzwingen eines linearen Zusammenhangs zwischen Abszissen- und Ordinaten-Werten ist nicht mehr zeitgemäß und wie ober diskutiert höchstens näherungsweise korrekt.

Numerische Minimierungsverfahren nutzen verschiedene, oft mehrschrittige Algorithmen zur Suche nach einem Minimum einer skalaren Funktion im K-dimensionalen Parameterraum. Solche Verfahren funktionieren sowohl für lineare als auch für nichtlineare Probleme, sind aber natürlich bei linearen Problemen rechenaufwändiger als das oben besprochene analytischen Lösungsverfahren. Nichtlineare Problemstellungen sind allerdings eher die Regel; auch ein zunächst lineares Problem kann durch Erweiterungen zur besseren Modellierung der Daten sehr schnell zu einem nichtlinearen werden. Bei nichtlinearen Problemen gibt es in der Regel mehr als ein Minimum, und ein Algorithmus findet nicht notwendigerweise das globale Minimum. Welches Minimum gefunden wird, hängt dann von den Startwerten und anfänglichen gewählten Schrittweiten ab, die solche Algorithmen grundsätzlich benötigen. Einer ersten groben Suche nach einem Minimum folgt üblicherweise eine zweite Stufe von effizienteren Algorithmen, die in der Nähe des vermuteten Minimums die ersten Ableitungen nach den Parametern nutzen, um die Konvergenz zu beschleunigen. Optional erlaubt z. B. das in *ROOT* zur Minimierung werwendete Paket *MINUIT* zur Steigerung der numerischen Effizienz, die Ableitungen der χ^2-Funktion nach den Parametern in Form vom Programmcode zu spezifizieren. Normalerweise werden die benötigten Ableitungen sowie die zweiten Ableitungen zur Konstruktion der Fehlermatrix numerisch bestimmt. Bei komplexen anzupassenden Funktionen kann es sogar notwendig werden, die Genauigkeit der Funktionsauswertung anzugeben, um numerisches Rauschen von einer tatsächlichen Änderung der χ^2-Funktion zu unterscheiden. Werden keine Angaben gemacht, so verwenden fast alle Programme vernünftige Standard-Werte, die in vielen Fällen zu guten Ergebnissen führen. Der *MINOS*-Algorithmus des *MINUIT*-Pakets ermöglicht die Bestimmung der Parameterunsicherheiten mittels eines Scans der Profil-Likelihood.

Es existieren eine Reihe von Programmen, die eine Funktionsanpassung mit numerischer Minimierung einer χ^2-Funktion bezüglich des Parametervektors $S(\boldsymbol{p})$ erlauben. Sie weisen z. T. starke Unterschiede in Bezug auf ihre Eigenschaften und Möglichkeiten auf. Bei den flexibelsten Programmpaketen mit eigenem Programmierinterface ist es möglich, die zu minimierende Funkion, sei es eine einfache oder an das Problem angepasste χ^2-Funktion oder eine problemspezifische Likelihood-Funktion, selbst anzugeben.

[2] Bei Experimenten aus dem Physikalischen Praktikum wären dies Größen wie Temperatur, Luftdruck, Luftfeuchte usw..

In den folgenden Abschnitten wird kurz auf einige Programme bzw. Softwarepakte eingegangen, die auf allen üblichen Platformen als offener Quellcode verfügbar sind.

A.1 Funktionsanpassung mit *qtiplot*

Das Programm *qtiplot* (http://wiki.ubuntuusers.de/qtiplot) ist in einigen Linux-Distributionen enthalten und frei verfügbar. Die Bedienung erfolgt über die grafische Oberfläche, die der Funktionalität des teuren *Origin* entspricht. Daten werden in Tabellenform eingegeben, wie man es aus Tabellenkalkulationen kennt, ein Datenimport aus ASCII-Dateien ist ebenfalls möglich, deren Format in Abbildung 6 zusammen mit einer typischen grafischen Darstellung des Ergebnisses gezeigt ist.

```
# x      y       dy
0,05     0,35    0,06
0,36     0,26    0,07
0,68     0,52    0,05
0,80     0,44    0,05
1,09     0,48    0,07
1,46     0,55    0,07
1,71     0,66    0,09
1,83     0,48    0,10
2,44     0,75    0,11
2,09     0,70    0,10
3,72     0,75    0,11
4,36     0,80    0,12
4,60     0,90    0,10
```

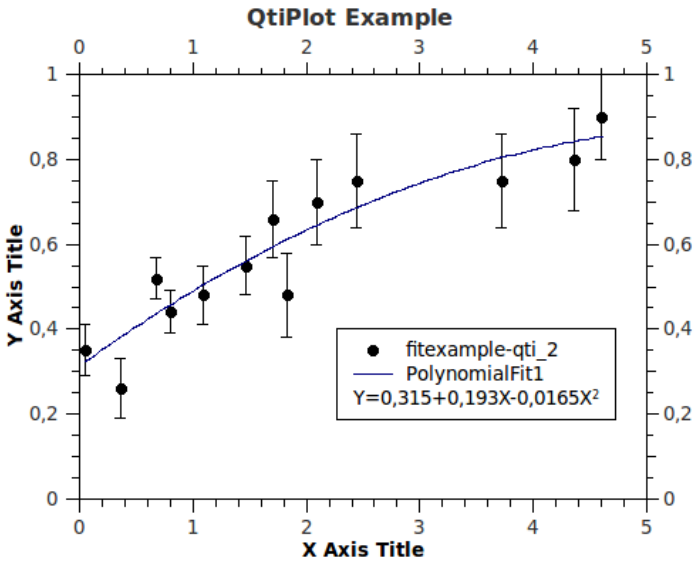

Abbildung 6: Beispiel für die Anpassung einer Parabel an Daten (links) mit *qtiplot*. Die Daten sind die gleichen wie in Abbildung 1, bei der Anpassung und grafische Ausgabe mit *RooFiLab* durchgeführt wurden.

Nach dem Starten des Programms werden über den Menü-Punkt File/Import/Import ASCII die Beispieldaten eingelesen und als Tabelle dargestellt. Die dritte Spalte muss nun mit der rechten Maustaste angeklickt werden, um im Kontext-Menü set as / Y Error anzuwählen. Als Standard für Anpassungen ist in *qtiplot* eine ungewichtete χ^2–Methode eingestellt; zur korrekten Berücksichtigung von Unsicherheiten der Eingabedaten müssen daher zunächst einige Optionen eingestellt werden. Die für die Anpassung vorgesehenen Felder in der Tabelle müssen mit der Maus markiert werden, dann den Menüpunkt Analysis/Fit Wizard ... wählen und die anzupassende Funktion festlegen. Im zweiten Reiter dieses Menüs können nun die Fit-Optionen eingestellt werden – zur Berücksichtigung der in der dritten Spalte der Tabelle angegebenen Unsicherheiten die Option Weighting: instrumental auswählen und dann die Aktion Fit zum Ausführen der Anpassung anklicken. Das Ergebnis der Anpassung einer Parabel an die Daten aus dem obigen Beispiel sieht dann wie folgt aus:

```
Polynomial Fit of dataset: fitexample-qti_2, using function: a0+a1*x+a2*x^2
Weighting Method: Instrumental, using error bars dataset: fitexample-qti_3
From x = 5,000e-02 to x = 4,600e+00
a0 = 3,155e-01 +/- 4,578e-02
```

```
a1 = 1,932e-01 +/- 5,905e-02
a2 = -1,652e-02 +/- 1,276e-02
--------------------------------------------------------------------------------
Chi^2/doF = 9,646e-01
R^2 = 0,864
Adjusted R^2 = 0,819
RMSE (Root Mean Squared Error) = 0,982
RSS (Residual Sum of Squares) = 9,65
```

Die Angabe `RSS` ist der χ^2-Wert der Anpassung, `RMSE` ist die Wurzel aus `{chi^2/doF}`, dem Wert von χ^2 dividiert durch die Zahl der Freiheitsgrade. Vorsicht: in den Standardeinstellungen werden die Unsicherheiten der Parameter mit diesem Wert skaliert, d. h. es wird angenommen, dass die angepasste Funktion die Daten genau beschreibt, χ^2/n_f also exakt Eins ist; das entspricht einer Skalierung alle Unsicherheiten der Eingabedaten mit dem gleichen Faktor. Dieses Verfahren wird auch angewandt, wenn keine Unsicherheiten angegeben werden. In diesem Fall sind die ausgegebenen Parameterfehler mit größter Vorsicht zu behandeln! Die Größe *(adjusted)* R^2 in der oben gezeigten Ausabe ist das sogenannte „(korrigierte) Bestimmtheitsmaß“, das die Übereinstimmung von Modell und Daten quantifiziert, allerdings die Unsicherheiten der Eingabedaten nicht berücksichtigt und daher in der Physik üblicherweise nicht verwendet wird.

qtiplot enthält eine ganze Reihe weiterer Möglichkeiten zur Darstellung und Analyse von Messdaten. Es sei an dieser Stelle auf die Online-Hilfe verwiesen.

A.2 Funktionsanpassung mit *gnuplot*

Das Programm *gnuplot* gibt es für Linux und Windows. Seine Hauptanwendung ist die Visualisierung von Daten und Funktionen, es beinhaltet aber auch die Möglichkeit, Funktionen an fehlerbehaftete Messdaten anzupassen.

Zur Anpassung einer Parabel an die in der Datei `fitexample.dat` im Format `" x  y  Fehler "` gespeicherten Messungen genügt in *gnuplot* die folgende einfache Befehlssequenz, die man auf der Kommandozeile nach dem Aufruf des Programms eingibt:

```
gnuplot> f(x) = a*x*x + m * x + b
gnuplot> fit f(x) 'fitexample.dat' using 1:2:3 via a,m,b
gnuplot> plot 'fitexample.dat' using 1:2:3 with errorbars ,f(x)
```

Man erhält damit die in Abbildung 7 gezeigte Grafik und folgende Ausgabe auf der Textkonsole:

```
degrees of freedom    (FIT_NDF)                       : 10
rms of residuals      (FIT_STDFIT) = sqrt(WSSR/ndf)    : 0.982122
variance of residuals (reduced chisquare) = WSSR/ndf   : 0.964564
Final set of parameters            Asymptotic Standard Error
=======================            ==========================
a               = -0.0165198       +/- 0.01253      (75.87%)
m               = 0.193232         +/- 0.058        (30.01%)
b               = 0.315463         +/- 0.04496      (14.25%)
correlation matrix of the fit parameters:
               a      m      b
a               1.000
m              -0.956  1.000
b               0.742 -0.855  1.000
```

```
# ------- data ------
# x       y      ey
 .05     0.35    0.06
0.36     0.26    0.07
0.68     0.52    0.05
0.80     0.44    0.05
1.09     0.48    0.07
1.46     0.55    0.07
1.71     0.66    0.09
1.83     0.48    0.1
2.44     0.75    0.11
2.09     0.70    0.1
3.72     0.75    0.11
4.36     0.80    0.12
4.60     0.90    0.1
```

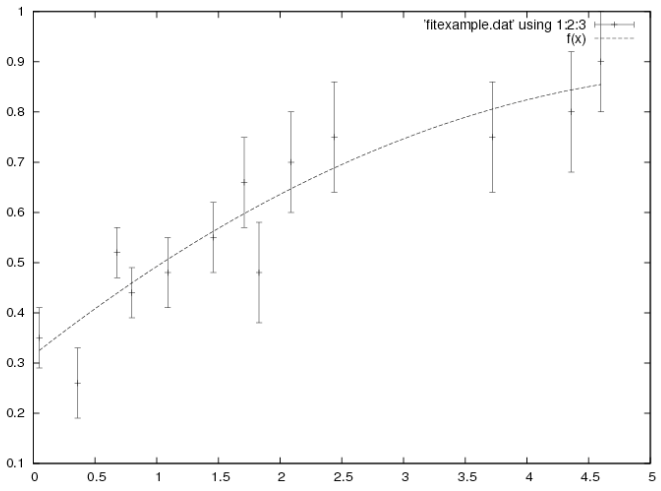

Abbildung 7: Beispiel für die Anpassung einer Parabel an Daten (links) mit *gnuplot*. Die Daten sind die gleichen wie in Abbildung 1, bei der Anpassung und grafische Ausgabe mit *RooFiLab* durchgeführt wurden.

ausgegeben werden nach erfolgreicher Anpassung die Zahl der Freiheitsgrade n_f der Anpassung, sowie der Wert von χ^2/n_f am Minimum (`WSSR`/ndf) und die Wurzel daraus. Außer den angepassten Parameterwerten und deren absoluten und relativen Unsicherheiten wird auch die Korrelationsmatrix der Parameter angezeigt. Im obigen Beispiel sind die Korrelationen sehr groß, es sollte also eine bessere Parametrisierung gewählt werden – etwa eine Verschiebung der x-Werte gemäß $x' = x - \overline{x}$.

A.3 Funktionsanpassung mit *ROOT*

Das Programmpaket *ROOT* ist ein mächtiges Software-Framework zur Datenanalyse für wissenschaftliche Problemstellungen. *ROOT* gibt es als vollständigen Quellcode oder vorcompiliert für Linux, und mit Einschränkungen auch für Windows. *ROOT* kann über eine Makro-Sprache in C++-Syntax oder über das Python-Interface *pyroot* interaktiv benutzt werden, eigener C++ code kann aber auch mit den Bibliotheken von *ROOT* gelinkt und so ein ausführbares Programm mit effizientem Maschinen-Code erzeugt werden.

ROOT enthält zwei Basisklassen, `TGraph` und `TH1`, die Methoden zur Anpassung von Funktionen bereit stellen. `TH1` ist einen Klasse zur Darstellung und Bearbeitung von Häufigkeitsverteilungen und daher nur in speziellen Fällen anwendbar. Die Klasse `TGraphErrors` stellt Messungen als fehlerbehaftete Datenpunkte (x_i, y_i) dar und erlaubt es, Anpassungen mit Unsicherheiten in Ordinaten- und Abszissenrichtung durchzuführen, allerdings ohne Korrelationen der Messfehler zu berücksichtigen. `TGraphErrors` bietet eine Anzahl von Methoden, Daten einzulesen und darzustellen und nutzt die Basisklasse `TVirtualFitter` zur Ausführung von Anpassungen. Komplexere Probleme lassen sich durch eigenen Programmcode bewältigen, der die Minimierung von beliebigen Nutzer-definierten Funktionen $S(\boldsymbol{p})$ mit Hilfe der durch zahlreiche Optionen konfigurierbaren Basisklasse `TVirtualFitter` durchführt.

A.3.1 Funktionsanpassung in Root: C++ – Makro

ROOT-Macros in C++ können interaktiv vom eingebauten Interpreter CINT ausgewertet, aber auch mit Hilfe des System-Compilers übersetzt und dann im Interpreter als schneller Maschinencode ausgeführt werden. Für

Details sei auf die Dokumentation von *Root* [3] verwiesen.

Der Konstruktor der Klasse `TGraphErrors` erlaubt die Übernahme von Daten aus einer Datei oder aus entsprechend initialisierten C-Arrays, wie im folgenden Beispiel gezeigt. Ist dieses *ROOT*-Macro in der Datei `example_TGraphFit.C` gespeichert, so lässt es sich nach dem Aufruf von *ROOT* direkt von der *ROOT*-Shell aus aufrufen:

```
root [0] .x example_TGraphFit.C
```

Das Beispiel demonstriert die Anpassung eines Polynoms 2. Grades an Daten mit Unsicherheiten in x- und y-Richtung.

```
// *** file example_TGraphFit.C ***
void TGraphFit() {
   //Draw a graph with error bars and fit a function to it.
   //Originial source Rene Brun    modified: Gunter Quast

  //set global options
  gStyle->SetOptFit(111);  //superimpose fit results

  // make nice Canvas
  TCanvas *c1 = new TCanvas("c1","Daten",200,10,700,500);
  c1->SetGrid();

  //define some data points
  const Int_t n = 10;
  Float_t x[n]  = {-0.22, 0.1, 0.25, 0.35, 0.5, 0.61, 0.7, 0.85, 0.89, 1.1};
  Float_t y[n]  = {0.7, 2.9, 5.6, 7.4, 9., 9.6, 8.7, 6.3, 4.5, 1.1};
  Float_t ey[n] = {.8,.7,.6,.5,.4,.4,.5,.6,.7,.8};
  Float_t ex[n] = {.05,.1,.07,.07,.04,.05,.06,.07,.08,.05};

  // copy data to TGraphErros object
  TGraphErrors *gr = new TGraphErrors(n,x,y,ex,ey);
  gr->SetTitle("TGraphErrors mit Fit");
  gr->Draw("AP");

  // now perform a fit(with errors in x and y!)
   gr->Fit("pol2");
   c1->Update();
}
```

Dieses Makro erzeugt Textausgaben im Konsolenfenster und die Grafik in einem separaten Grafikfenster, das interaktive Möglichkeiten zur Bearbeitung und zum Export der Grafik anbietet. Auch der Export als Makro in C++-Syntax ist möglich - auf diese Art kann man Beispiele für die Anwendung der Methoden der unterschiedlichen Grafik-Klassen für die Darstellung von Rahmen, Achsen, Text usw. erhalten.

A.3.2 Funktionsanpassung in Root: Python–Makro

Wenn die *Python*-Erweiterung *pyroot* zusammen mit dem *Root*-Paket installiert ist und der entsprechende Pfad zu den Python-Scripten von *Root*, `ROOTSYS/lib`, in der Umgebungsvariablen `PYTHONPATH` eingetragen ist, lassen sich *Root*–Klassen auch aus der weit verbreiteten, objektorientierten Script-Sprache *Python* verwenden. Details und ein einführendes Tutorial zu *Python* findet man unter `http://docs.python.org/2/tutorial/`.

[3] `http://root.cern.ch/`

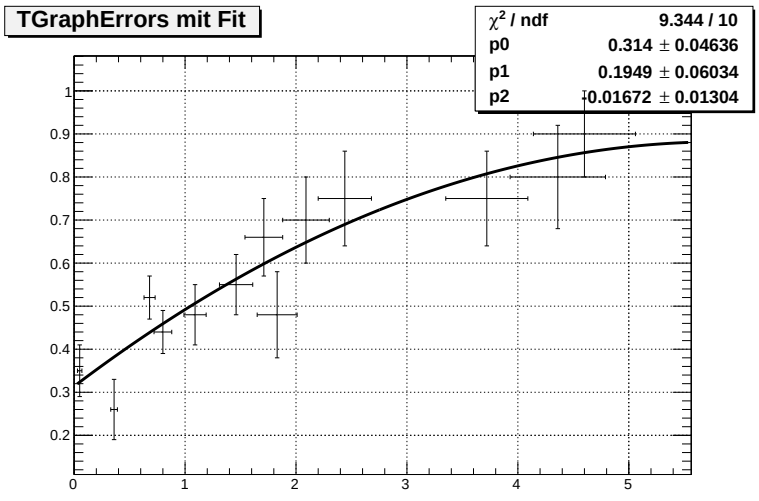

Abbildung 8: Grafische Ausgabe des *ROOT*-Makros.

Das unten angegebene Beispiel-Script, das genau dem C++-Makro aus dem letzten Abschnitt entspricht, lässt sich einfach vom Betriebssystem aus starten – dafür sorgt die erste Zeile. Das Script kann auch mit
`python example_TGraphFit.py`
oder auch in der interaktiven und sehr empfehlenswerten Shell *ipython* mit
`ipython example_TGraphFit.py`
gestartet werden, auch wenn die erste Zeile fehlt.

```
# *** file example_TGraphFit.py ***
#!/usr/bin/env python
# ---------------------------------------------------------
# Draw a graph with error bars and fit a function to it

from ROOT import gStyle, TCanvas, TGraphErrors
from array import array

gStyle.SetOptFit(111)  # superimpose fit results
c1=TCanvas("c1","Daten",200,10,700,500) #make nice Canvas
c1.SetGrid()

#define some data points ...
x = array('f',(.05,0.36,0.68,0.80,1.09,1.46,1.71,1.83,2.44,2.09,3.72,4.36,4.60))
y = array('f',(0.35,0.26,0.52,0.44,0.48,0.55,0.66,0.48,0.75,0.70,0.75,0.80,0.90))
ey = array('f',(0.06,0.07,0.05,0.05,0.07,0.07,0.09,0.1,0.11,0.1,0.11,0.12,0.1))
ex = array('f',(0.02,0.03,0.05,0.08,0.1,0.15,0.17,0.18,0.24,0.21,0.37,0.43,0.46))
nPoints=len(x)
# ... and pass to TGraphErros object
gr=TGraphErrors(nPoints,x,y,ex,ey)
gr.SetTitle("TGraphErrors mit Fit")
gr.Draw("AP");
# now perform a fit (with errors in x and y!)
```

```
gr.Fit("pol2")
c1.Update()
# request user action before ending (and deleting graphics window)
raw_input('Press <ret> to end -> ')
```

A.4 Funktionsanpassung mit dem Python-Paket *kafe*

Einen Schritt weiter bei der Kapselung der *Root*-Klassen geht das universelle und einfach zu steuernde und in der Sprache *Python* geschriebene Paket *kafe* [10], das in Karlsruhe entwickelt wurde. Es erlaubt Anpassungen mit der *Root*-Klasse `TMinuit` und nutzt für numerische Rechnungen und zur grafischen Darstellung die *Python*-Pakete *numpy*, *scipy* und *matplotlib*.

Der komplette Arbeitsablauf wird von einem kleinen, in der Sprache *Python* geschriebenen Script gesteuert. Hier ein Beispiel:

```
from kafe import *
from kafe.function_tools import FitFunction, LaTeX, ASCII
from kafe.function_library import quadratic_3par

# fit function definition with decorators for nice output
@ASCII(expression='a * x^2 + b * x + c')
@LaTeX(name='f', parameter_names=('a', 'b', 'c'), expression='a\\,x^2+b\\,x+c')
@FitFunction
def poly2(x, a=1.0, b=0.0, c=0.0):
    return a * x**2 + b * x + c

# set fit function
fitf=poly2             # own definition
#fitf=quadratic_3par   # or from kafe function library

# Workflow #
############## load the experimental data from a file
my_dataset = parse_column_data(
    'kafe_fitexample.dat',
    field_order='x,y,xabserr,yabserr',
    title='example data')
# Create the Fit
my_fit = Fit(my_dataset, fitf)
# Do the Fits
my_fit.do_fit()
# Create the plots
my_plot = Plot(my_fit)
# Set the axis labels
my_plot.axis_labels = ['$x$', 'data \& $f(x)$']
# Draw the plots
my_plot.plot_all()

# Plot output #
###############
# Save the plots
my_plot.save('kafe_fitexample.pdf')
```

```
# Show the plots
my_plot.show()
```

Das Eingabeformat für die Daten ist sehr flexibel, die Ausgabe in grafischer Form sehr zeitgemäß und ansprechend. Die Definition der Fit-Funktion nutzt die sogenannten „Decorator Functions“ von *Python*, um neben dem Code für die Berechnung optional auch eine Funktionsdarstellung in LaTeX sowie aussagekräftige Namen für die Parameter der Fit-Funktion angeben zu können. Außer der im Beispiel gezeigten Definition im Code stehen im Modul `kafe.function_library` eine ganze Reihe an vordefinierten Fit-Funktionen zur Verfügung.

Abbildung 9 zeigt die Ausgabe des obigen Scripts für den ebenfalls gezeigten Test-Datensatz.

```
# x    y    ex   ey
0.05 0.35 0.02 0.06
0.36 0.26 0.03 0.07
0.68 0.52 0.05 0.05
0.80 0.44 0.08 0.05
1.09 0.48 0.1  0.07
1.46 0.55 0.15 0.07
1.71 0.66 0.17 0.09
1.83 0.48 0.18 0.1
2.44 0.75 0.24 0.11
2.09 0.70 0.21 0.1
3.72 0.75 0.37 0.11
4.36 0.80 0.43 0.12
4.60 0.90 0.46 0.1
```

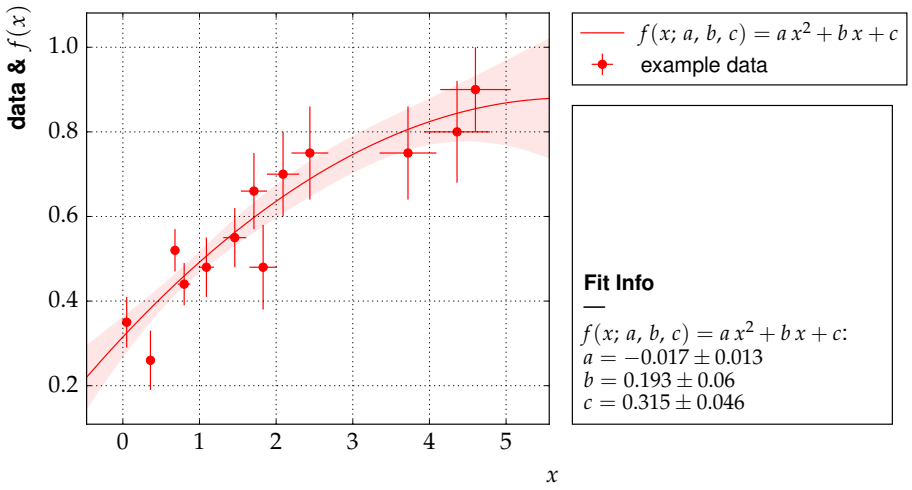

Abbildung 9: Beispiel für die Anpassung einer Parabel an die Beispieldaten mit *kafe*.

kafe kann mit korrelierten Unsicherheiten auf Ordinate und Abszisse umgehen und stellt die Unsicherheit der angepassten Kurve als Konfidenzband (bei 68 % Konfidenzniveau entsprechend $\pm 1\,\sigma$) um den Zentralwert der angepassten Funktion dar. Das Band wird unter Berücksichtigung der Korrelationen der angepassten Parameter mittels Fehlerfortpflanzung berechnet.

Weitere Funktionen erlauben einen Scan der χ^2-Funktion in der Nähe des Minimums und die Bestimmung von Konfidenzintervallen und Konfidenz-Konturen, die bei stark nicht-linearen Problemen sinnvoller sind als die Angabe von (symmetrischen) Unsicherheiten und Korrelationskoeffizienten.

Das installierbare *Python*-Paket mit Quellcode, Dokumentation und zahleichen Beispielen findet sich unter `http://www.ekp.kit.edu/~quast/kafe`.

Da der volle Funktonsumfang der Programmiersprache *Python* mit vielen Hilfspaketen und auch direkter Zugriff auf die im Paket *kafe* implementierten Methoden der im obigen Beispiel genutzten Klassen zur Verfügung stehen, ist die Funktionsanpassung sehr flexibel und erlaubt auch die Behandlung komplexer Anwendugnsfälle. Für die Datenauswertung in Praktika und Bachelor- oder Masterarbeiten sei an dieser Stelle das Paktet *kafe* nachdrücklich empfohlen. Die mitgelieferten einfachen und nur aus ganz wenigen Zeilen Programmcode bestehenden Beispiele decken den größten Teil der praktisch vorkommenden Typen von Problemen ab. Komplexere oder sehr spezielle Problemstellungen lassen sich Dank der Erweiterbarkeit durch eigenen Code bzw. Hinzunahme weiterer *Pyhton*-Bibliotheken oder von Funktionalität des *ROOT*-Pakets behandeln.

A.5 Funktionsanpassung mit *RooFiLab*

Obwohl im Prinzip einfach, stellt die Formulierung eines Problems in C++ und die komplexe Umgebung des *ROOT*-Frameworks eine relativ hohe Hürde dar. Eine Vereinfachung und Erweiterung der Funktionalität bringt das Programm *RooFiLab*, dass dem Anwender für die meisten einfachen Fälle die Erstellung von eigenem C++- oder *P*ythonCode erspart.

Basierend auf *ROOT* stellt das in Karlsruhe entwickelte Programm *RooFiLab* eine Erweiterung zur Durchführung von Parameter-Anpassungen auf der Basis der χ^2-Methode zur Verfügung. Die Anwendung für den Einsteiger wird durch eine strukturierte grafische Oberfläche erleichtert.

In den in *ROOT* direkt oder in Programmpaketen wie z. B. *gnuplot* verfügbaren Anpassungsmethoden werden Kovarianzmatrizen nicht berücksichtigt. *RooFiLab* erweitert *ROOT* um diese Funktionalität und stellt eine vereinfachte, strukturierte grafische Benutzeroberfläche zur Verfügung, mit deren Hilfe Funktionsanpassungen an Messwerte unter Berücksichtigung korrelierter Unsicherheiten der Ordinate und der Abszisse möglich werden. In allgemeinen Fall können kompliziertere Fehlermodelle durch das Einlesen von explizit angegebenen Kovarianzmatrizen in der Anpassung verwendet werden. Für den einfachen Fall gemeinsamer absoluter oder relativer Unsicherheiten aller Messwerte enthält das Programm eine vereinfachte Eingabemöglichkeit. Ein Beispiel ist in Abbildung 10 gezeigt.

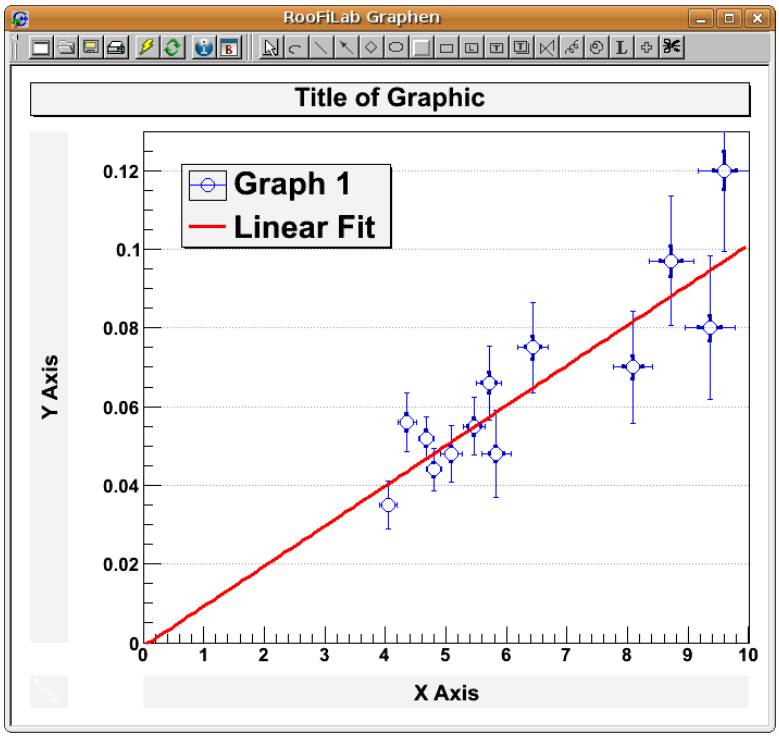

Abbildung 10: Beispiel einer Geradenanpassung mit (korrelierten) systematischen Unsicherheiten in x- und y-Richtung.

Eine hohe Flexibilität bei der Definition der anzupassenden Funktion wird einerseits durch die Interpreter-Funktion von *ROOT* erreicht, die die Eingabe von Funktionen in einer einfachen Formelsprache erlaubt. Daneben können aber auch komplexere, als C- oder C++-Code implementierte Funktionen zur Laufzeit des Programms eingebunden werden.

Die Elemente der grafischen Oberfläche des Programms *RooFiLab* und die Steuerung über die Eingabedatei sind im Handbuch `RooFiLab.pdf` im Verzeichnis `RooFiLab/doc` genau beschrieben, hier folgt nur ein kurzer Überblick.

A.5.1 Installation

RooFiLab wird komplett installiert in einer virtuellen Maschine auf Ubuntu-Basis [4] bereit gestellt. Das gepackte Festplattenabbild kann mit Hilfe des frei verfügbaren Virtualisierers *VirtualBox* auf gängigen Linux-Distributionen, Windows-Versionen ab Windows XP und Macintosh-Computern mit Betriebssystem Mac OS X ausgeführt werden (siehe URL http://www.virtualbox.org sowie gesonderte Anleitung und Hilfe-Datei zur Virtuellen Maschine, `http://www.ekp.kit.edu/~quast`).

Der Programm-Code von *RooFiLab* findet sich unter der gleichen URL in der gepackten Datei `RooFiLab.tar.gz`. Unter Linux ist damit die Installation mit Hilfe der im Unterverzeichnis `RooFiLab` enthaltenen Quelldateien möglich. Die Datei `Makefile` enthält alle notwendigen Anweisungen zur Erzeugung der ausführbaren Datei durch einfaches Aufrufen von `make`. Dazu muss eine *ROOT*-Installation vorhanden und initialisiert sein, d. h. die Umgebungsvariable `PATH` muss den Pfad zu den ausführbaren *ROOT*-Dateien und die Umgebungsvariable `LD_LIBRARY_PATH` den Pfad zu den *ROOT*-Bibliotheken enthalten.

A.5.2 Bedienung des Programms *RooFiLab*

RooFiLab bietet dem Anwender eine in zwei Programmfester untergliederte Oberfläche: Im rechten (Haupt-) Fenster erfolgt die Steuerung des Programms, während in dem anderen Fenster die entsprechende Grafik dargestellt wird.

Die Steuerung ist in vier Shutter untergliedert. Die Eingaben bzw. Aktionen der jeweiligen Shutter,

- Einlesen der Daten und Definition der Fit-Parameter,
- Festlegen sinnvoller Anfangswerte sowie „Fit-by-Eye“,
- Durchführen der Anpassung, ggf. mit Fixierung einzelner Fit-Parameter,
- Festlegen von Optionen und Bearbeitung der Ausgabe-Grafik

sollten nacheinander ausgeführt werden.

Zur Ausführungszeit stehen einige Funktionen von *ROOT* zur Verfügung, insbesondere für die grafische Darstellung, die interaktive Manipulation und den Export der erstellten Grafiken. Dies wird durch öffnen der Kontext-Menüs von *ROOT* durch Rechtsklicks in die entsprechenden Komponenten der Grafik und mit Hilfe der Toolbar im oberen Graphikfenster ermöglicht.

Neben der Nutzung der Steuerelemente der grafischen Oberfläche können Anpassungen auch automatisiert durch die Angabe von Steueroptionen in der Eingabedatei ausgeführt werden, in der auch die Datenpunkte definiert werden. Die zunächst in einer interaktiven Anpassung ermittelten Eingabeoptionen können so in der Eingabedatei archiviert und für wiederholte, automatisierte Anpassungen verwendet werden.

[4] `http://www.ekp.kit.edu/~quast/VMroot`

Literatur

[1] G. Bohm, G. Zech, *Einführung in Statistik und Messdatenanalyse für Physiker*, DESY eBook, `http://www-library.desy.de/preparch/books/vstatmp.pdf`

[2] V. Blobel u. E. Lohrmann, *Statitische Methoden der Datenanalyse*, Teubner sowie `http://www.desy.de/~blobel/eBuch.pdf`

[3] R. Barlow, *Statistics*, Wiley

[4] G. Cowen, *Statistical data analysis*, Oxford Science Publications

[5] S. Brandt, *Datenanalyse*, Spektrum akademischer Verlag

[6] W.H. Press, *Numerical Recipes in C++*, Cambridge University Press

[7] Data Analysis Framework ROOT; *Root Users Guide*, `http://root.cern.ch` ROOT beginners guide: *Diving into Root*, D. Piparo, G. Quast, `http://www.ekp.kit.edu/~quast`

[8] Programm *qtiplot*, Home Page `http://soft.proindependent.com/qtiplot.html`, Information zu freien Version unter Linux siehe `http://wiki.ubuntuusers.de/qtiplot`

[9] Programm *gnuplot*, Home Page `http://www.gnuplot.info`

[10] D. Savoiou, Bachelorarbeit EKP 2013 (IEKP-BACHELOR-KA-2013-19, http://ekp-invenio.physik.uni-karlsruhe.de/record/48310/)

[11] Home Page G. Quast, dieses und weitere Scripte, Virtuelle Maschine, RooFiLab, Python-Paket kafe `http://www.ekp.kit.edu/~quast`