Script: Anpassen von Funktionen an Messdaten

21. September 2022

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 in Abszissenrichtung zusätzlich zu denen 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.etp.kit.edu/~quast`
E-Mail: `G.Quast@kit.edu`

Inhaltsverzeichnis

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 Unsicherheiten in Richtung der Ordinaten-Achse behaftete Messpunkte angepasst wurde.

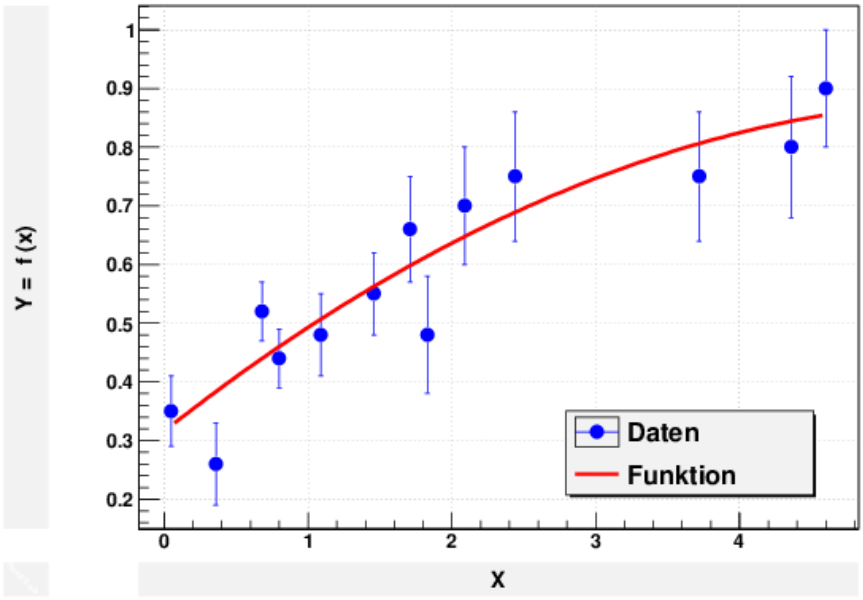

Abbildung 1: Beispiel der Anpassung einer Funktion (hier $f(x) = ax^2 + bx + c$) an Messpunkte (x_i, y_i).

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, das 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) implementiert sind. Es gibt auch eine Reihe quelloffene und frei verfügbare Bibliotheken mit darauf aufbauenden Anwendungen. Einige dieser Möglichkeiten und einfache praktische Beispiele werden im Anhang A etwas näher behandelt.

2 χ^2-Methode zur Funktionsanpassung

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

Häufig wird zur Funktionsanpassung an Datenpunkte mit Unsicherheiten 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 Messunsicherheiten. Als Abstandsmaß verwendet man in diesem Fall ein Skalarprodukt mit Gewichten, die dem Inversen der quadrierten Messunsicherheiten 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 residuals, WSSR“). Für gaußförmig um den wahren Wert verteilte Messunsicherheiten folgt der Wert S_0 on S am Minimum bzgl. der Parameter 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, die Verteilungsdichte von S_0 ist also $\mathrm{f}(S_0) = \chi^2(S_0; N-K)$. Daher hat sich für diese Methode auch der Name „χ^2-Methode“ etabliert.

Geschrieben mit den Messunsicherheiten 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 $f(x_i)$ gaußförmig verteilte Messwerte y_i ist die χ^2-Methode äquivalent zum Likelihood-Verfahren, s. Kap. 6.1.

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 Verteilungen der Messunsicherheiten existieren.

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

1. zunächst erfolgt eine Anpassung ohne Berücksichtigung der Unsicherheiten der Abszissenwerte;
2. im zweiten Schritt werden diese dann mit Hilfe der ersten Ableitungen $f'(x_i)$ der im ersten Schritt angepassten Funktion f in entsprechende Unsicherheiten der Ordinate umgerechnet und quadratisch addiert: ${\sigma_i}^2 = {\sigma_{y_i}}^2 + (f'(x_i) \cdot \sigma_{xi})^2$. Die Größe χ^2 wird jetzt mit den neuen Unsicherheiten σ_i berechnet und minimiert.
3. 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.

Dank der in den letzten Jahren ständig gestiegenen Rechenleistung ist es auch möglich, die Unsicherheiten im zweiten Schritt während des numerischen Minimierungsprozesses dynamisch anzupassen. Nach der Konvergenz zum Minimum entfällt in diesem Fall der dritte Schritt.

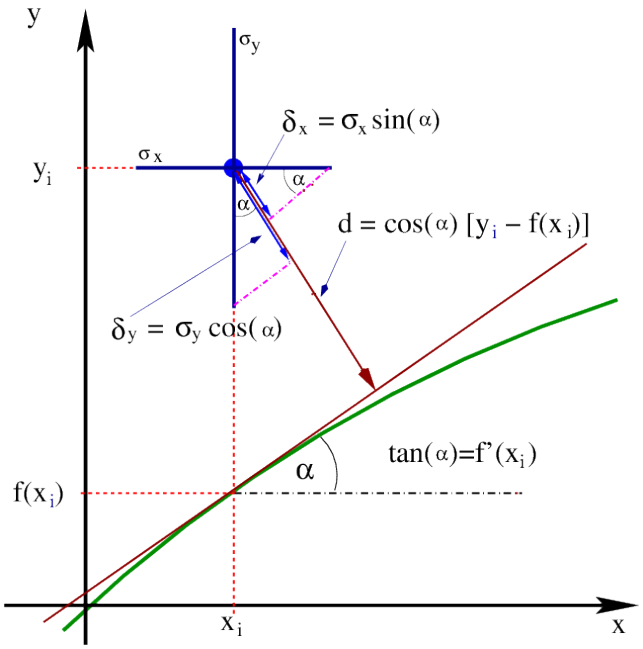

Abbildung 2: Illustration des Abstandsmaßes für einen Punkt mit Unsicherheiten in Richtung der Ordinaten- und Abszissenachse. Das im Text beschriebene Verfahren entspricht der Minimierung des normierten Abstands der Messpunkte $(x_i,\, y_i)$ von der Tangente durch den Punkt $(x_i,\, f(x_i))$

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 die Messunsicherheit 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 Messunsicherheiten von x_i ist, wird dieses Verfahren ungenau.

2.1 Behandlung von korrelierten Unsicherheiten

Unsicherheiten von Messwerten sind typischerweise einerseits bestimmt durch unabhängige Unsicherheiten jeder Einzelmessung, wie z.B. Ablesefehler, statistische Unsicherheiten usw.. Andererseits gibt es systematische Unsicherheiten, die alle Messwerte in gleicher Weise betreffen, also zwischen diesen „korreliert“ sind. Die Beschreibung solcher korrelierter Unsicherheiten 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 Gesamtunsicherheiten $(\sigma^{(t)})$ der Messwerte, $C_{ii} = {\sigma_i^{(t)}}^2$; die Nebendiagonalelemente enthalten die gemeinsamen Komponenten $(\sigma_i^{(g)})$ der Unsicherheiten, $C_{ij} = \sigma_i^{(g)} \cdot \sigma_j^{(g)}$ der Messungen mit den Indizes i und j.

Wenn die Messunsicherheiten korreliert sind, wird χ^2 mit Hilfe der Kovarianzmatrix $\mathbf{C}$ 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^x_{ij} beschrieben und mit Hilfe der 1. Ableitung zu den Kovarianzmatrixelementen C^y_{ij} der Ordinatenwerte 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.

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 Verfahren lässt sich mit praktisch allen Programmen zur Anpassung von Modellfunktionen an Daten iterativ implementieren. Wenn das Miminmierungspaket Zugriff auf das zu minimierende Abstandsmaß zulässt, kann die Kovarianzmatrix auch in jedem Iterationsschritt der numerischen Optimierung dynamisch neu berechnet werden.

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:

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

Werden mehrere Parameter angepasst, so sind deren Unsicherheiten häufig korreliert, selbst wenn die 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 Unsicherheiten der Datenpunkte angegeben, so werden in der χ^2-Summe alle Gewichte, d. h. die Unsicherheiten 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 Parameterunsicherheiten 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 Parameterunsicherheiten 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 Unsicherheiten vermutet, so führt man zunächst eine Anpassung mit $\sigma_i = 1$ durch und erhält χ^2/n_f. Die von der Anpassung gelieferten Unsicherheiten 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 Unsicherheiten 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 gaußförmiger Verteilung der Unsicherheiten 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:

$$\chi^2{}_{\text{prob}} = \int_{\chi_{min}}^{\infty} \chi^2(s; n_f)\,\mathrm{d}s \;=\; 1. - \int_{0}^{\chi_{min}} \chi^2(s; n_f)\,\mathrm{d}s \quad {}^{1}. \tag{4}$$

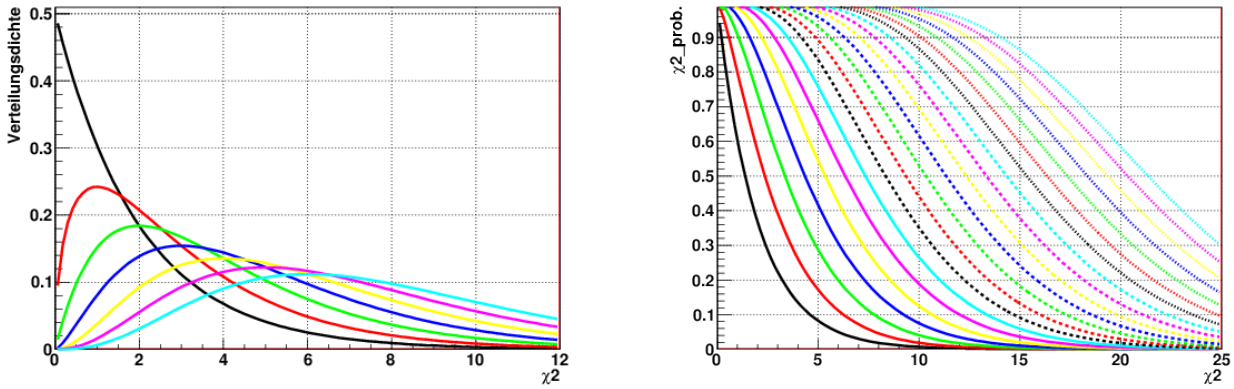

Abbildung 3: χ^2-Verteilung (links) für 2 – 8 Freiheitsgrade und χ^2-Wahrscheinlichkeit für 2 – 22 Freiheitsgrade (rechts). Der Erwartungswert der χ^2-Verteilung ist n_f und ihre Standardabweichung ist $\sqrt{2n_f}$. Für eine sehr große Anzahl von Freiheitsgraden geht die Verteilung in eine Gaußverteilung 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.

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

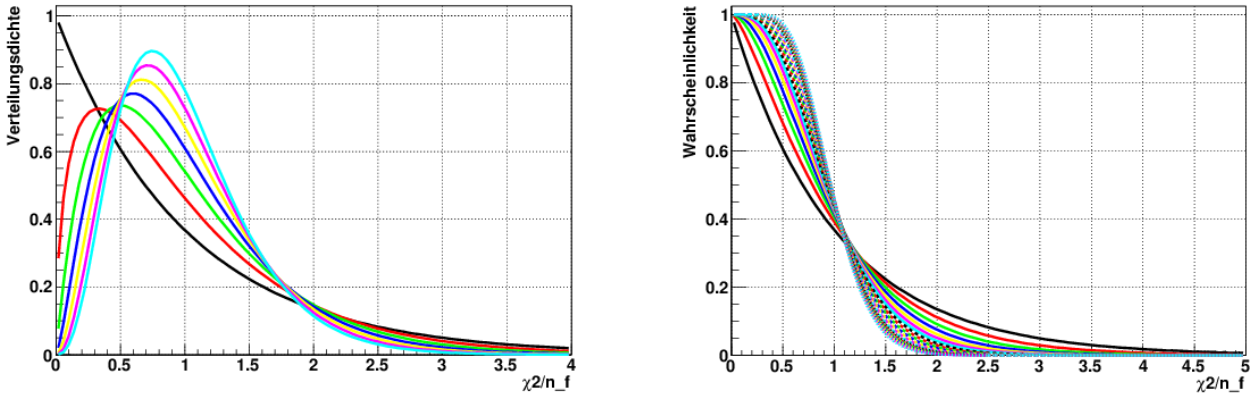

Abbildung 4: Verteilungsdichte von χ^2/n_f (links) für 2 – 8 Freiheitsgrade und die entsprechende Wahrscheinlichkeit für 2 – 22 Freiheitsgrade, einen größeren Wert als den auf der x-Achse angegebenen zu finden (rechts).

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 die Gesamtunsicherheiten der Messwerte y_i gegeben:

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

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

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

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

Wenn es mehrere korrelierte Einzelkomponenten gibt, so erhält man die gesamte Kovarianzmatrix durch Addition aller so berechneten Kovarianzmatrizen. Dies entspricht der quadratischen Addition von Einzelunsicherheiten – 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 $-1 \leq Cor_{ij} \leq 1$. Ist Cor_{ij} Eins, so sind die Messungen y_i und y_j vollständig 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 Gesamtunsicherheiten 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 $\mathbf{A}$ 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 Messunsicherheiten. 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 Messunsicherheiten 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

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

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

$$\hat{p}'_1 = \frac{S_y}{S_1} = \overline{y}\,, \qquad {\sigma_{p'_1}}^2 = \frac{1}{S_1}\,,$$
$$\hat{p}'_2 = \frac{S_{x'y}}{S_{x'x'}} = \frac{\overline{x'y}}{\overline{x'^2}}\,, \qquad {\sigma_{p'_2}}^2 = \frac{1}{S_{x'x'}}\,.$$

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 Unsicherheiten der Messwerte nicht berücksichtigt, d. h. $\sigma_i = 1$ für alle i und damit $S_1 = N$. Das hier beschriebene Verfahren mit Berücksichtigung von Messunsicherheiten 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 eingesetzt. 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.

4.2 Mittelwertbildung von korrelierten Messungen

Eine weitere sehr wichtige Anwendung der Ergebnisse in Gleichungen 6 und 7 ist die Mittelung korrelierter Messergebnisse. Die Mittelwertbildung kann man auffassen als eine Anpassung an eine konstante Funktion, $f(x_i) = \bar{m}$, so dass alle Messwerte innerhalb der Messunsicherheiten dem gleichen Mittelwert $\bar{m}$ entsprechen sollten.

In diesem Fall wird die Matrix A in Gl.5 zu einem Vektor: $A = (1, \ldots, 1)^T$, und man erhält durch Einsetzen in Gl. 6 das Ergebnis:

$$\bar{m} = \frac{1}{\sum_{i,j} (W)_{i,j}} \sum_{i,j} (W)_{i,j}\, x_j \tag{8}$$

Für die Varianz von $\bar{m}$ ergibt sich:

$$V_{\bar{m}} = \frac{1}{\sum_{i,j} (W)_{i,j}} \tag{9}$$

Dabei ist W wie schon oben die Inverse der Kovarianzmatrix der Messungen y_i.

Das hier vorgestellte Verfahren ist die beste unverzerrte Schätzung des Mittelwerts, wenn die Messwerte Gaußverteilt sind. Unterschiedlich große Unsicherheiten der Messwerte und damit einher gehende unterschiedliche Gewichte im Mittelungsprozess werden korrekt berücksichtigt.

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 Unsicherheiten 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 Quadrate der Unsicherheiten 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 einer kleineren angenommenen Unsicherheit, 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 aus den beobachteten Werten berechneten Unsicherheiten durchgeführt,
– im zweiten Schritt werden die Unsicherheiten 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 der exakten Verteilungen der Unsicherheiten erforderlich. Dann sind die Grenzen der Anwendbarkeit der χ^2-Methode erreicht.

6 Die Likelihood-Methode

Als Alternative zum oben eingeführten Abstandsmaß von Messdaten und Modellfunktion bietet sich das Likelihood-Verfahren an, das 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 (natürlichen) Logarithmus der Likelihood, lässt konstante, d. h. nicht vom Parametervektor abhängige Terme weg und erhält die „negative Log-Likelihood Funktion“ für das Problem,

$$-\ln\,\mathcal{L}(\vec{n};\boldsymbol{p}) \;=\; \sum_{i=1}^{N} -n_i \cdot \ln(\mu_i(\boldsymbol{p})) \;+\; \mu_i(\boldsymbol{p})\,, \tag{10}$$

die man bzgl. der Parameter minimiert.

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

$$V_{ij}{}^{-1} = \left. \frac{\partial^2 \ln \mathcal{L}(\vec{n}; \boldsymbol{p})}{\partial p_i \, \partial p_j} \right|_{\hat{p}_i \hat{p}_j} . \tag{11}$$

Die Anwendung des Liklihood-Verfahrens wird notwendig, wenn die Verteilung der Unsicherheiten stark von der Gaußverteilung abweicht oder 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 Addition der den beobachteten Messwerten entsprechenden logLikelihood-Werte die Gesamt-logLikelihood des Problems bestimmt werden:

$$- \ln \mathcal{L} = - \sum_{i=1}^{N} \ln \left(\, \mathcal{P}_i(x_i, y_i; \, \boldsymbol{p}) \, \right) \tag{12}$$

6.1 Likelihood und χ^2-Methode

Für n Datenpunkte $(\boldsymbol{x}, \boldsymbol{y})$ mit um die Modellwerte $f(\boldsymbol{x}; \boldsymbol{p})$ **gaußförmig verteilten Unsicherheiten mit Kovarianzmatrix** C ist der negative Logarithmus der Gaußverteilung gegeben durch

$$- 2 \, \ln \mathcal{L}_{\text{Gauß}}(\boldsymbol{x}; \boldsymbol{\mu}, C) \, = \, (\boldsymbol{y} - \boldsymbol{f}(\boldsymbol{x}; \boldsymbol{p}))^T \, C^{-1} \, (\boldsymbol{y} - \boldsymbol{f}(\boldsymbol{x}; \boldsymbol{p})) + \ln(\det(C)) + n \, \ln(2\pi) \, . \tag{13}$$

Die nicht von den Parametern abhängenden Summanden kann man weglassen, da die Lage des Minimums in Parameterraum nicht davon abhängt. Wenn es nur feste, d. h. paramterunabhängige Unsicherheiten gibt, entspricht $- \ln \mathcal{L}$ also bis auf einen Faktor $\frac{1}{2}$ der χ^2-Größe,

$$- 2 \ln \mathcal{L}_{\text{Gauß}} \, = \, (\boldsymbol{y} - \boldsymbol{f}(\boldsymbol{x}; \boldsymbol{p})^T \, C^{-1} \, (\boldsymbol{y} - \boldsymbol{f}(\boldsymbol{x}; \boldsymbol{p}) \, = \, \chi^2 \, . \tag{14}$$

Wenn relative Unsicherheiten bezüglich des Modellwerts oder Unsicherheiten in Abszissenrichtung berücksichtigt werden sollen, hängt, wie wir oben gesehen hatten, die Kovarianzmatrix C von den Parameterwerten ab, $C(f(\boldsymbol{x}; \boldsymbol{p}))$. Der Summand $\ln(\det(C))$ mit der Determinanten der Kovarianzmatrix sollte dann nicht weggelassen, sondern in der zu minimierenden Kostenfunktion berücksichtigt werden:

$$- 2 \ln \mathcal{L}_{\text{Gauß}} \, = \, \chi^2 \, (f(\boldsymbol{x}, \boldsymbol{p})) + \ln \left(\det(C(f(\boldsymbol{x}, \boldsymbol{p}))) \right) . \tag{15}$$

Bei der numerischen Bestimmung der besten Parameterwerte wird diese Kostenfunktion in jedem Iterationsschritt dynamisch angepasst. Da dazu die Inverse und die Determinante der parameterabhängigen Kovarianzmatrix benötigt werden, ist das Verfahren numerisch aufwändig, auf modernen Computern mit zeitgemäßen numerischen Bibliotheken aber kein wirkliches Problem. Allerdings skaliert der Rechenaufwand stark nichtlinear mit der Dimension der Kovarianzmatrix, also der Anzahl an Datenpunkten.

Zur Bewertung der Qualität der Anpassung kann weiterhin die Größe χ^2 verwendet werden. Dies scheint intuitiv richtig, kann aber auch unter Verwendung der Likelihood begründet werden: Das bestmögliche Modell würde die Datenpunkte exakt beschreiben, alle Residuen $(\boldsymbol{y} - \boldsymbol{f}(\boldsymbol{x}; \boldsymbol{p}))$ wären Null, und der bestmögliche Wert der Kostenfunktion ist daher $\ln \det(C)$. χ^2 ist also der Unterschied im Logarithmus der Likelihood zwischen der bestmöglichen und der tatsächlich beobachteten Übereinstimmung zwischen Daten und Modellfunktion, eine „log-Likelihood-Differenz“ ähnlich der, die wir weiter unten (s. Abschnitt 7) zur Bestimmung von Vertrauensintervallen nutzen werden.

Abschließend bleibt festzuhalten, dass die Maximierung der Likelihood, oder, völlig äquivalent, die Minimierung des negativen Logarithmus der Likelihood, in der Regel aufwändige numerische Verfahren erfordert, da im Allgemeinen $-\ln\mathcal{L}$ eine komplizierte Funktion der n Datenpunkte und der K Parameter ist. Allgemeine Software-Implementierungen können daher kaum vorgenommen werden, sondern es muss auf das jeweilige Problem angepasster, eigener Programmcode erstellt werden.

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. Ein Konfidenzintervall, das dem Bereich $[\mu-\sigma,\mu+\sigma]$ eine Gaußverteilung, also dem zentralen 68 % Quantil entspricht, 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 Bestimmung der Unsicherheiten mit Hilfe der zweiten Ableitungen am Minimum nach Formel 11. Wegen der Äquivalenz von χ^2- und Likelihood-Methode für gaußförmige Unsicherheiten der Eingabedaten nach Formel 14 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 auftreten, wie in Abbildung 5 am Beispiel der Anpassung einer Exponentialfunktion illustriert ist.

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 Werte zu überprüfen und ggf. durch die Grenzen des aus dem Scan der Profil-Likelihood gewonnen Intervalls zu ersetzen.

8 Abschließende Anmerkungen

Die χ^2-Methode ist ein Spezialfall eines Liklelihood-Verfahrens für Gauß-verteilte Unsicherheiten der Eingangsgrößen. Wenn die Voraussetzung gaußförmiger 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 gaußförmige Unsicherheiten 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

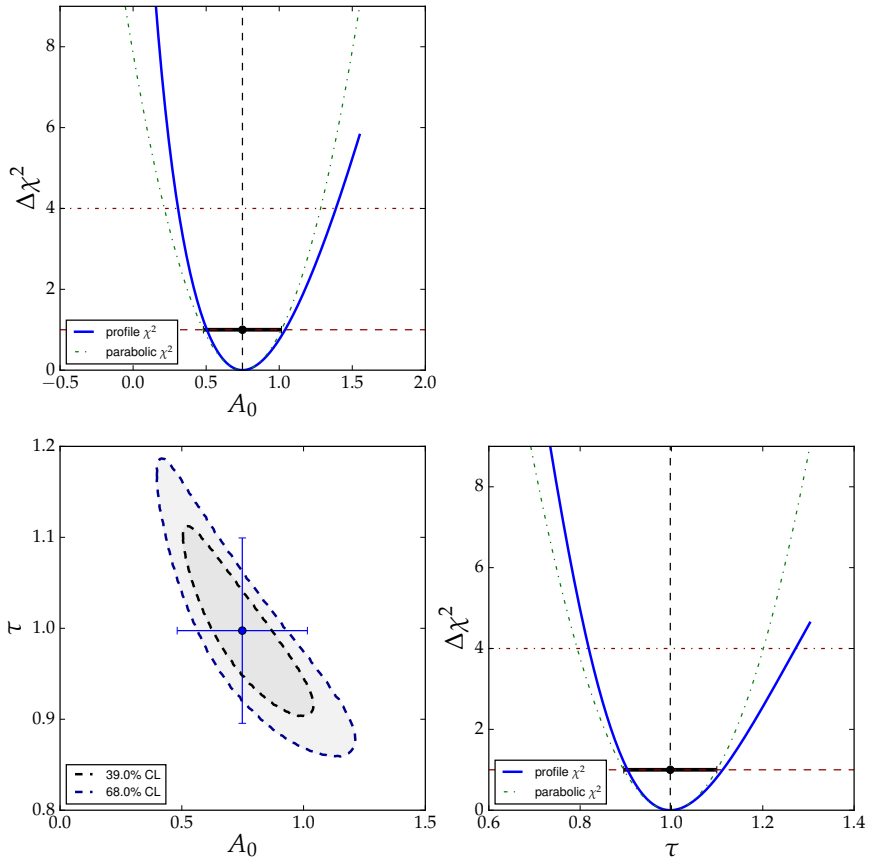

Abbildung 5: Beispiel der Anpassung einer Exponentialfunktion $A(t; A_0, \tau) = A_0 \exp\left(\frac{-t}{\tau}\right)$ (erzeugt mit dem unten beschriebenen Paket *kafe*). Für die beiden Parameter der Funktion ist der Verlauf der Profil-Likelihood gezeigt, der mit den aus den Krümmungen am Minimum bestimmten Parabeln verglichen wird. Die im Bild links unten gezeigte, aus der Profil-Likelihood gewonnene Konfidenz-Kontur weicht ebenfalls sehr stark von der Ellipsenform 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.

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 Unsicherheiten 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 parameterabhängigen Likelihood-Funktionen werden von Physikern selbst programmiert und die Optimierung und statistische Interpretation unter Zuhilfenahme von Standardbibliotheken ausgeführt. Einige der unten aufgeführten, auf den Programmiersprachen C++ oder *Python* beruhende Programmpakete erlauben es, solche problemspezifischen Erweiterungen einzubauen.

A Anhang: 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

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

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. Auch die Berücksichtiung von Unsicherheiten in Abszissen-Richtung oder von relativen, auf den Modellwert statt auf die gemessenen Datenpunkte bezogene relative Unsicherheiten stellt kein grundsätzliches Problem mehr dar.

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. *MINIUT* steht auch als gekapseltes Paket *iminuit* für die Sprache Python zur Verfügung. Normalerweise werden bei der Computer-basierten Optimierung 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 Bibliotheken oder ausführbaren Programmen („Apps“ in moderner Sprechweise), 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 und Zugriff auf die verwendete Kostenfunktion ist es möglich, die zu minimierende Funkion, die sog. „Kostenfunktion“, frei zu definieren. Damit sind angefangen von einer einfachen oder auch speziell an das Problem angepassten χ^2-Funktion bis hin zu einer problemspezifischen negativen Log-Likelihood-Funktion alle Möglichkeiten gegeben.

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.

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

# 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

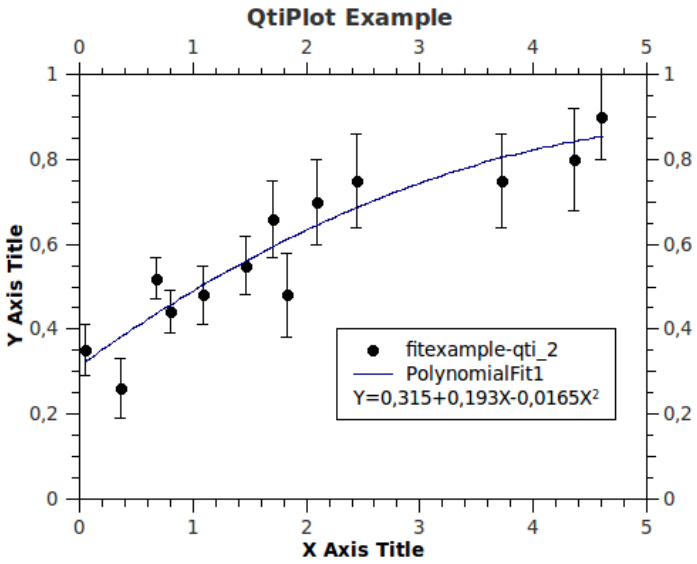

Abbildung 6: Beispiel für die Anpassung einer Parabel an Daten (links) mit *qtiplot*. Die Daten sind die gleichen wie in Abbildung 1.

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, das die angepasste Funktion die Daten genau beschreibt, χ^2/n_f also exakt Eins ist, und alle Unsicherheiten der Eingabedaten werden mit dem gleichen Faktor skaliert. Dieses Verfahren wird auch angewandt, wenn keine Unsicherheiten angegeben werden. In diesem Fall sind die ausgegebenen Parameterunsicherheiten mit größter Vorsicht zu behandeln!

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* ist der Klassiker, den es für alle Plattformen als freie Software gibt. Seine Hauptanwendung ist zwar die Visualisierung von Daten und Funktionen, aber *gnuplot* beinhaltet aber auch die Möglichkeit, Funktionen an mit Unsicherheiten behaftete Messdaten anzupassen.

Zur Anpassung einer Parabel an die in der Datei `fitexample.dat` im Format `" x  y  sigma_y "` 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:

```
# ------- 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
```

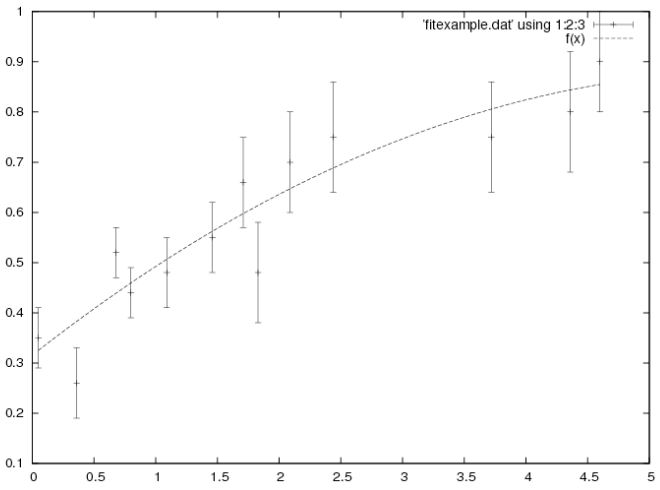

Abbildung 7: Beispiel für die Anpassung einer Parabel an Daten (links) mit *gnuplot*. Die Daten sind die gleichen wie in Abbildung 1.

```
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
```

Angezeigt werden die Zahl der Freiheitsgrade, n_f, der Anpassung, sowie der Wert von χ^2/n_f am Minimum (WSSR/ndf). Auch die Korrelationsmatrix der Parameter wird mit ausgegeben. Im obigen Beispiel sind die Korrelationen sehr groß - eine bessere Parametrisierung sollte gewählt werden – etwa eine Verschiebung der x-Werte gemäß $x' = x - \overline{x}$.

A.3 Funktionsanpassung mit Python-Skripten

Für die Programmiersprache Python gibt es eine Reihe von frei verfügbaren Optimierern, d. h. Algorithmen, die eine numerische Optimierung einer skalaren, von Parametern abhängigen Kostenfunktion durchführen. Zu nennen sind hier insbesondere die in der Python-Bibliothek *scipy.optimize* enthaltenen Algorithmen, oder auch das auf dem am europäischen Zentrum für Teilchenphysik, CERN in Genf, entwickelten Minimierer *MINUIT* beruhende Python-Paket *iminuit*.

Damit lassen sich sehr mächtige und vielseitige, flexible Verfahren zur Anpassung von Modellen an Messdaten implementieren. Oft sind dazu nur wenige eigene Programmzeilen notwendig. Beispiele dazu sind in der für die physikalischen Praktika bereitgestellten Sammlung von Python-Skripten und Werkzeugen *PhyPraKit*[10] enthalten.

Sehr einfach und häufig ausreichend ist das Paket *s*cipy.optimize.curve_fit, das allerdings lediglich unabhängige Unsicherheiten der Ordinate berücksichtigen kann. Der *Python*-Code zur Durchführung des schon oben verwendete Standardbeispiels sieht folgendermaßen aus:

```
import numpy as np
from scipy.optimize import curve_fit

# fit function definition
def poly2(x, a=1.0, b=0.0, c=0.0):
    return a * x**2 + b * x + c

#1. load data
x, y, sy = np.loadtxt("fitexample.dat", unpack=True)
# linear least squares with scipy.optimize.curve_fit
par, cov = curve_fit( poly2, x, y, sigma=sy, absolute_sigma=True )
print("Fit parameters:\n", par)
print("Covariance matrix:\n", cov)
```

Das Script liefert folgende Ausgabe in Textform:

```
Fit parameters:
 [-0.01651975  0.19323224  0.31546257]
Covariance matrix:
 [[ 0.00016287 -0.00072041  0.0004335 ]
 [-0.00072041  0.0034873  -0.00231134]
 [ 0.0004335  -0.00231134  0.00209543]]
```

Die Option `absolute_sigma=True` sorgt dafür, dass die angegebenen Unsicherheiten nicht skaliert werden, um einen Wert von $\chi^2/n_{dof} = 1$ zu erzwingen. Dieses Verhalten ist auch in diesem Fall die Voreinstellung, die aktiv überschrieben werden muss.

Sollen alle acht in diesem Skript besprochenen Arten von Unsicherheiten, also unabhängige und/oder korrelierte absolute und/oder relative Unsicherheiten in x- und/oder y-Richtung, berücksichtigt werden, ist die Auswahl an

fertigen Lösungen gering. Deshalb muss auf speziell angepasste Implementierungen auf Basis flexibler Optimierer zurückgegriffen werden.

Eine mit minimalistischem Aufwand erstelle Lösung bietet die Eigenimplementierung *phyFit*, die auf *iminuit* beruht und die Kostenfunktion aus Gl. 15 verwendet. Dieser Beispielcode soll als Vorlage für eigene Entwicklungen dienen und ist auch zum Austesten von Erweiterungen gedacht. Der Code ist trotz des aufwändigen Algorithmus recht schnell und enthält eine einfache Implementierung grafischer Darstellungen der Daten mit überlagerter Modellfunktion und von Profile-Likelihood-Kurven und Kovarianzkonturen. Die Unsicherheit der angepassten Kurve wird als Konfidenzband (mit 68 % Konfidenzniveau entsprechend $\pm 1\,\sigma$) um den Zentralwert der angepassten Funktion dargestellt. Dieses Band wird unter Berücksichtigung der Korrelationen der angepassten Parameter mittels Fehlerfortpflanzung aus der Modellfunktion berechnet.

Für die meisten der im Grundstudium auftretenden Probleme ist diese Lösung ausreichend. Der folgende *Python*-Code illustriert die Anwendung von *phyFit*.

```
''' general example for fitting with iminuit
      - read datafile fitexample.dat
      - perform fit (2nd order polynomial)
      - show output
'''

### import everything we need
from PhyPraKit.phyFit import mnFit
import numpy as np, matplotlib.pyplot as plt

### fit function definition
def fitf(x, a=1.0, b=0.0, c=0.0):
    return a * x**2 + b * x + c

### Workflow #
# 1. load data
x, y, sy = np.loadtxt("fitexample.dat", unpack=True)
# 2. set up Fit object
myFit = mnFit()
# 3. initialize data object and uncertainties
myFit.init_data(x, y, ey=sy, erelx=0.05)
# 4. initalize fit object
myFit.init_fit(fitf)
# 5. perform fit
myFit.do_fit()
# 6. retrieve results
FitResult = myFit.getResult()
# 7. make result plot
fig = myFit.plotModel()
# 8. print results
np.set_printoptions(precision=3)
print("*===* Result of fit:\n", FitResult)
# 9.finally, save figure and/or show plot
## plt.savefig('iminuit_fitexample.pdf')
plt.show()
```

Die mit diesen Programmzeilen erzeugte grafische Ausgabe ist in Abbildung 8 dargestellt.

```
# x    y    ex    ey
0.05 0.35 0.003 0.06
0.36 0.26 0.018 0.07
0.68 0.52 0.034 0.05
0.80 0.44 0.040 0.05
1.09 0.48 0.055 0.07
1.46 0.55 0.073 0.07
1.71 0.66 0.086 0.09
1.83 0.48 0.092 0.10
2.44 0.75 0.12  0.11
2.09 0.70 0.10  0.10
3.72 0.75 0.19  0.11
4.36 0.80 0.22  0.12
4.60 0.90 0.23  0.10
```

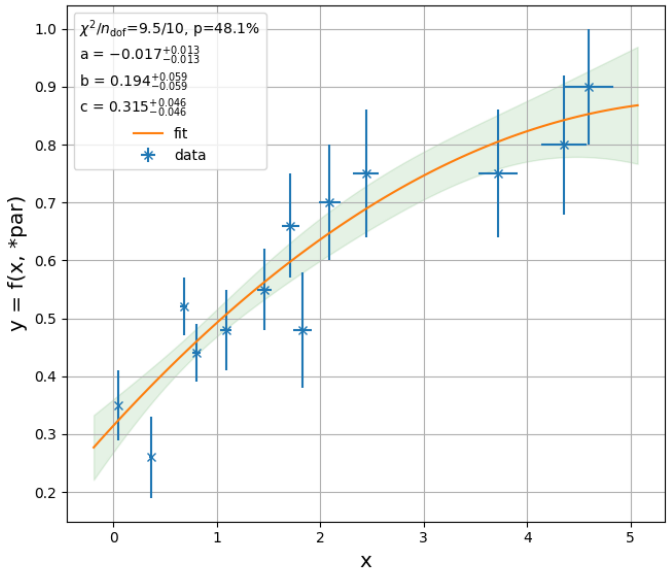

Abbildung 8: Beispiel für die Anpassung einer Parabel an die Beispieldaten mit *iminuit*. Zur Illustration der besonderen Fähigkeiten wurde eine relative Unsicherheit von 5 % in x-Richtung hinzugefügt.

Im Beispielordner `examples/` von von *PhyPraKit* findet sich auch das nützliche Skript *run_phyFit.py*, mit dem sich Anpassungen sehr komfortabel ohne die Erstellung von eigenem Python-Code durchführen lassen. Die Daten sowie das anzupassende Modell werden in diesem Fall aus einer in der Datenbeschreibungssprache *yaml* erstellen Datei gelesen. Die Anpassung wir dann einfach mittels Eingabe von

```
> python3 run_phyFit.py fit.yml
```

auf der Kommandozeile ausgeführt. Das Format der Eingabedatei ist weiter unten als Beispiel zur Anwendung von *kafe2go* erklärt.

A.4 Funktionsanpassung mit dem Python-Paket *kafe / kafe2*

Das universelle in *Python* geschriebene Paket *kafe* [11], das in Karlsruhe entwickelt wurde und mittlerweile in der 2. Version als *kafe2* [12] vorliegt, kapselt eine ganze Reihe an Funktionen und numerische Optimierern. *kafe2* erlaubt Anpassungen mit den Python-Paketen *iminuit* oder *scipy.optimize* oder auch mit der *ROOT*-Klasse `TMinuit` (s. u.) und nutzt für numerische Rechnungen und zur grafischen Darstellung die *Python*-Pakete *numpy*, *scipy* und *matplotlib*.

Neben der Möglichkeit der Durchführung einer Anpassung mit einem Python-Skript ermöglicht auch *kafe2* die Verwendung einer Datenbeschreibungssprache („yaml“), um einfache Anpassungen auszuführen, ohne eigenen Code zu schreiben (s. weiter unten). Die Steuerung über ein *Python*-Skript erfordert aber auch nur wenige Zeilen. Hier ein Beispiel, das die einzelnen Schritte einer Anpassung mit *kafe2* zeigt:

```
''' general example for fitting with kafe2
      - read datafile fitexample.dat
      - perform fit (2nd order polynomial)
      - show and save output
'''
# Imports  #
from kafe2 import XYContainer, Fit, Plot
import numpy as np, matplotlib.pyplot as plt

### define the model function
def poly2(x, a=1.0, b=0.0, c=0.0):
    return a * x**2 + b * x + c

### Workflow #
# 1. load the experimental data from a file
x, y, e = np.loadtxt('fitexample.dat', unpack=True)

# 2.  convert to kafe2 data structure and add uncertainties
xy_data = XYContainer(x, y)
xy_data.add_error('y', e)                    # independent errors y
xy_data.add_error('x', 0.05, relative=True)  # independent relative errors x
# --  set meaningful names
xy_data.label = 'Beispieldaten'
xy_data.axis_labels = ['x', 'data & f(x)']

# 3. create the Fit object
my_fit = Fit(xy_data, poly2)
# set meaningful names for model
my_fit.model_label = 'Parabel-Modell'

# 4. perform the fit
my_fit.do_fit()

# 5. report fit results
my_fit.report()

# 6. create and draw plots
my_plot = Plot(my_fit)
my_plot.plot()

# 7. show or save plots #
## plt.savefig('kafe_fitexample.pdf')
plt.show()
```

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

Das Eingabeformat für die Daten und deren Unsicherheiten ist sehr flexibel. *kafe* kann mit allen hier diskutierten Arten von Unsicherheiten umgehen und enthält umfangreiche Ausgabeformate der Fit-Ergebnisse zur Weiterverwendung in Programmen und in Form von Text und Grafiken. 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. Die grafischer Ausgabe ist in weiten Grenzen konfigurierbar.

```
# x     y     ex    ey
.05   0.35  0.003 0.06
0.36  0.26  0.018 0.07
0.68  0.52  0.034 0.05
0.80  0.44  0.040 0.05
1.09  0.48  0.055 0.07
1.46  0.55  0.073 0.07
1.71  0.66  0.086 0.09
1.83  0.48  0.092 0.10
2.44  0.75  0.12  0.11
2.09  0.70  0.10  0.10
3.72  0.75  0.19  0.11
4.36  0.80  0.22  0.12
4.60  0.90  0.23  0.10
```

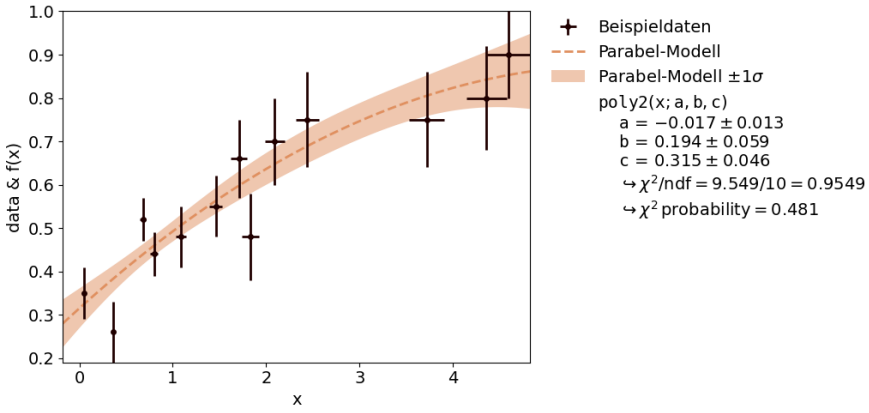

Abbildung 9: Beispiel für die Anpassung einer Parabel an die Beispieldaten mit *kafe2*. Hinzugefügt wurde noch eine relative Unsicherheit von 5 % in x-Richtung.

Das installierbare *Python*-Paket mit Quellcode, Dokumentation und zahlreichen Beispielen findet sich in einem Repository auf *github.com*, `https://github.com/PhiLFitters/kafe2` oder auf der Homepage `http://www.etp.kit.edu/~quast/kafe`.

Da der volle Funktionsumfang der Programmiersprache *Python* mit vielen Hilfspaketen und auch direkter Zugriff auf die im Paket *kafe2* implementierten Methoden der im obigen Beispiel genutzten Klassen zur Verfügung stehen, ist die Funktionsanpassung sehr flexibel und erlaubt auch die Behandlung komplexer Anwendungsfälle. Für die Datenauswertung in Praktika und Bachelor- oder Masterarbeiten sei an dieser Stelle das Paktet *kafe2* 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 *Python*-Bibliotheken behandeln. Zur einfacheren Einarbeitung enthält das Paket *kafe2* ein Tutorial als *jupyter* Notebook [3].

A.4.1 Anpassung mit *kafe2go*

Das Paket *kafe2* enthält das eigenständige Skript *kafe2go.py*, mit dem Anpassungen ganz ohne die Notwendigkeit der Erstellung von eigenem Python-Code durchgeführt werden können. Die Daten und die Modellfunktion sind in einer Datei im Format der einfachen Datenbeschreibungssprache *yaml* enthalten, wie unten im Beispiel gezeigt. Der Aufruf erfolgt mittels

```
> kafe2go kafe2go_fit.yml
```

auf der Kommandozeile.

Die ausführliche Dokumentation des *kafe2*-Pakets enthält auch einen Abschnitt zur Anwendung von *kafe2go* mit der Dokumentation der entsprechenden Schlüsselwörter zur Dateneingabe und zur Steuerung der Anpassung. Die Möglichkeiten sind im Vergleich zur Verwendung von Python-Code etwas eingeschränkter, dafür ist die Durchführung von Anpassungen aber sehr einfach. Für MS-Windows gibt es eine ausführbare Datei, die

[3] Zu *jupyter* s. Abschnitt A.6

Anpassungen mit Hilfe eines Rechtsklicks auf eine Datei im *yaml*-Format und anschließendem Öffnen mit der Anwendung *kafe2go.exe* ermöglicht.

Die *yaml*-Datei für unser Stadardbeispiel, die Anpassung einer Parabel an Datenpunkte, ist unten gezeigt.

```
# fit.yml: example of a fit with kafe2go
# ----------------------------------------------
#  execute via:
#    kafe2go fit.yml

# In this example a 2nd order polynomial
# is fit to data points with uncertainties.

# ----------------------------------------------

label: 'Beispieldaten'

x_data: [.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]
x_errors: 5%
x_label: 'x'

y_data: [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]
y_errors: [.06,.07,.05,.05,.07,.07,.09,.1,.11,.1,.11,.12,.1]
y_label: 'data & f(x)'

model_label: 'Quadratic model'
model_function: |
  def poly2(x, a=0., b=1., c=0.):
      # Our first model is a simple linear function
      return a * x*x + b*x + c
```

Abbildung 10: Beispiel für die Anpassung einer Parabel an die Beispieldaten mit *kafe2go*, das exakt dem Beispiel mit *kafe2* von eben entspricht und auch das identische Ergebnis liefert.

A.5 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 Messunsicherheiten 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.

ROOT-Macros in C++ können interaktiv vom eingebauten Interpreter ausgewertet, aber auch mit Hilfe des System-Compilers übersetzt und dann als schneller Maschinencode ausgeführt werden. Für Details sei auf die Dokumentation von *ROOT* [4] verwiesen.

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 mit *Python* verwenden.

A.6 Tutorials als *jupyter* Notebooks

In letzter Zeit haben sich Server-basierte Oberflächen für Anwendungen im Bereich der Visualisierung und Auswertung von Daten aller Art etabliert - Stichworte wie *Big Data* oder *Data Science* ziehen sich durch die Presse.

Das quelloffene Projekt *jupyter* [13] bietet eine Web-basierte interaktive Umgebung für wissenschaftliche Berechnungen und Datenauswertung. Die zu Grunde liegenden Dateien sind sogenannte „Notebooks“, die sowohl formatierten Text als auch Programmcode (auch) in der Sprache *Python* und die entsprechende Programmausgabe in Form von Text und Grafiken in einem einzigen Dokument vom Typ *.ipynb* speichern.

Vorhandene *Python*-Skripte sind direkt oder mit sehr wenig Aufwand in der *jupyter*-Umgebung lauffähig. Zur Formatierung eingegebener Texte wird eine leicht zu erlernende Markup-Sprache verwendet, die auch Ausdrücke in LaTeXkorrekt verarbeitet. Die Konversion vorhandener Textbausteine ist also leicht zu bewerkstelligen. Notebooks können in verschiedenen Formaten als komplette Dokumente, insb. auch *.pdf* und *.html*, exportiert und ggf. ausgedruckt werden.

Zur Anzeige und Bearbeitung dient ein beliebiger Web-Browser, der sich mit einem *jupyter*-Server verbindet. Ein solcher Server kann entweder lokal auf dem gleichen System wie der Browser laufen, oder aber auch auf einem über das Netzwerk angebundenen Rechner bereit gestellt werden. Bei Verwendung eines Servers im Netzwerk oder auch in einer Compute-Cloud muss das Gerät des Endnutzers nicht besonders leistungsfähig sein, sondern lediglich einen der gängigen Web-Browser unterstützten.

Als *jupyter*-Notebooks erstellte Tutorials zu Grundlagen der Statistik, zur Fehlerrechnung und Modellanpassung in Praktika finden sich auf der Homepage [14].

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 *R*OOT; *ROOT Users Guide*, `http://root.cern.ch` ROOT beginners guide: *Diving into ROOT*, D. Piparo, G. Quast, `http://www.etp.kit.edu/~quast`

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

[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] G. Quast, Programmsammlung *PhyPraKit*, Dokumentation unter `http://www.etp.kit.edu/~quast/PhyPraKit/htmldoc/`

[11] D. Savoiou, Bachelorarbeit EKP 2013 (IEKP-BACHELOR-KA-2013-19)

[12] *kafe2*, Projekt auf *github.com*: `https://github.com/dsavoiu/kafe2`

[13] *jupyter* Projektseite unter dem Link `https://jupyter.org/`

[14] Home Page G. Quast mit Material zu Grundlagen der Statistik, zur Fehlerrechnung und Modellanpassung, mit Tutorials als *jupyter*-Notebooks, `http://www.etp.kit.edu/~quast`