Jannick Weißhaupt
Franziska Flegel

Matlab für das Grundpraktikum

Inhaltsverzeichnis

1 Einführung in Matlab

Matlab (MATrix LABoratory) ist ein Programm und gleichzeitig eine Programmiersprache zur Lösung numerischer Probleme. Dabei ist es, wie der Name schon sagt, vor allem auf die Verwendung von Matrizen ausgelegt. Es kann mit einer Vielzahl von numerischen Problemen umgehen, wie z.B. Matrixalgebra, Differentialgleichungen, Integration, partielle Differentialgleichungen usw. und wird deshalb vor allem für die numerische Simulation von Problemen und zur Datenanalyse eingesetzt. Die Oberfläche von Matlab bietet einen Editor an, in dem Skripte, Funktionen, Klassen usw. geschrieben werden können. Befehle können auch direkt in das „Command prompt“ eingegeben werden, wo auch Ausgabetext erscheint. Außerdem hilfreich ist der „Workspace“, in dem die Variablen, die gerade im Speicher sind, eingesehen werden können. Bei Problemen sei auch auf die umfangreiche und verständliche Hilfe hingewiesen.

1.1 Darstellung einer Zahl in Matlab

In Computern können stets nur Zahlen mit endlich vielen Stellen dargestellt werden. Mehr Stellen bringen dabei natürlich eine höhere Genauigkeit, verbrauchen allerdings auch mehr Speicherplatz und Rechenzeit bei mathematischen Operationen. Normalerweise benutzt man sogenannte 64-Bit floating point numbers, welche 64 Bit Speicherplatz brauchen. Da Computer intern stets mit den Zuständen 1 („an“) und 0 („aus“) arbeiteten, eignet sich die binäre Darstellung am besten für Computer. Das Prinzip einer floating point number ist nun, dass die Größenordnung durch den Exponent dargestellt wird und die Zahl somit stets mit einer 1 beginnt. Die Nachkommastellen werden dann mit 52 Bit dargestellt. Ein Bit wird für das Vorzeichen verwendet und 11 Bit für den Exponenten. Durch den Exponenten wird ein Zahlenraum von $2^{2^{10}} \approx 10^{-308}$ bis $2^{2^{10}} \approx 10^{308}$ abgedeckt. Die relative Genauigkeit ist durch die Mantisse gegeben und beträgt $2^{-52} \approx 10^{-16}$ Ein Beispiel für die floating point number 43 mit 5 Bit Mantisse und 4 Bit Exponent lautet:

$$(\underbrace{-}_{\text{sign}} 1, \underbrace{01011}_{\text{Mantisse}} \times (10) \underbrace{^{+101}}_{\text{Exponent}})_2 = (-(1 + 2^{-2} + 2^{-4} + 2^{-5}) \times 2^5)_{10} = (43)_{10}$$

1.2 Definition von Variablen

In Programmiersprachen wird eine Variable üblicherweise von rechts nach links definiert. D.h. der Ausdruck, der rechts vom Gleichheitszeichen steht, wird ausgewertet und das Ergebnis unter der Variable, die links vom = steht gespeichert. In vielen

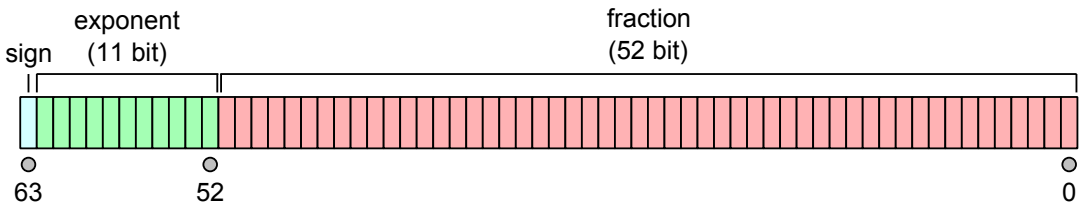

Abbildung 1.1.1: Speicheraufteilung einer 64-Bit floating point number.
Quelle: http://commons.wikimedia.org/wiki/File:General_double_precision_float.png

Sprachen muss die Variable zusätzlich vordefiniert werden, was in Matlab allerdings nicht nötig ist.

1.2.1 Skalare

Skalare werden in Matlab einfach durch

$$\text{Variable} = (\text{Zahl o. Rechenausdruck})$$

definiert. Dabei sind auch mathematisch sinnlose Gleichungen wie $b = b+1$ möglich. Zunächst wird dann der Rechenausdruck rechts ausgewertet. Das Ergebnis wird dann in b gespeichert, d.h. in diesem Fall wird b überschrieben. Als Ergebnis erhalten wir also, dass b um eins erhöht wird. Andere Beispiele:
$a = 5$, $b = 5 * 4$, $z = 1 + 1i$

1.2.2 Matrizen

Eine Matrix wird durch Skalare die mit [ ] eingeklammert sind erzeugt. Spalten werden durch Leerzeichen getrennt, Zeilenumbruch durch ; erzeugt.
Bsp: $a = [1\ 5\ 3; 2\ 5\ 1]$ ergibt $a = \begin{pmatrix} 1 & 5 & 3 \\ 2 & 5 & 1 \end{pmatrix}$
Die Elemente können dann mit $a(n, m)$ aufgerufen werden. Eine ganze Zeile oder Spalte ruft man mit $a(:, m)$ bzw. $a(n, :)$ auf.

Besondere Matrizen:

eye(n): $n \times n$ Einheitsmatrix.

zeros(n,m): $n \times m$ Nullmatrix.

ones(n,m): $n \times m$ Einsmatrix.

rand(n,m): $n \times m$ Matrix mit gleichverteilten Zufallszahlen.

randn(n,m): $n \times m$ Matrix mit standardnormalverteilten Zufallszahlen.

1.2.3 Anonymous Functions

Anonymous Functions stellen einfache Funktionen dar, die als Eingabewert einen oder mehrere Skalare bzw. Matrizen benötigen. Sie sind vor allem nützlich um einfache Funktionen an Routinen zu übergeben, welche als Eingabewert eine Funktion benötigen, wie z.B. Nullstellensuche, Differentialgleichungslöser oder Fitroutinen. Die Syntax lautet z.B.

$$\text{func1} = @(x)\ x\text{^}2 + \sin(x * \text{pi})$$

oder ein anderes Beispiel für mehrere Eingabewerte

$$\text{func2} = @(x1, x2)\sqrt{(x1 - x2)' * (x1 - x2)}$$

Aufgerufen werden sie durch z.B.

$$\text{func1}(2) \Rightarrow 4$$

und als Beispiel für Funktionen mit mehreren Eingabewerten

$$\text{func2}([1; 0; 0], [0; 0; 1]) \Rightarrow 1.4142$$

1.2.4 Cells

Cells funktionieren ähnlich wie Matrizen. Sie unterscheiden sich allerdings darin, dass ihre Elemente quasi alles sein können, z.B Strings, Zahlen, Matrizen, Anonymous Functions usw. Sie werden mit {...} erzeugt, statt wie bei Matrizen mit [...]. Ansonsten ist die Syntax gleich; Leerzeichen für Spaltenwechsel und ; für Zeilenwechsel.
Bsp.: cell1 = $\{@(x)x, @(x)x.\text{^}2\}$ Ein Element der Cell kann durch `cell{index}` aufgerufen, also z.B. durch

$$\text{func2} = \text{cell1}\{2\}.$$

Dabei wird das zweite Element der Cell, also hier die Funktion x^2, aufgerufen und unter der Variablen func2 abgespeichert.

1.3 Einfache Rechenoperationen

Einfache Operationen können durch

$$\text{Variable} = \text{Rechenbefehl}$$

durchgeführt werden. Das Ergebnis wird dann in der Variablen abgespeichert. Dabei müssen natürlich alle im Rechenbefehl verwendeten Variablen vorher schon definiert worden sein.

1.3.1 Addition und Subtraktion

Addition und Subtraktion werden mit + und − durchgeführt. Dabei können natürlich sowohl Skalare als auch Matrizen gleicher Größe miteinander addiert werden. Addiert man zwei Matrizen unterschiedlicher Größe, gibt Matlab eine Fehlermeldung aus. Als einzige Ausnahme kann zu einer Matrix ein Skalar addiert werden. Dabei wird dann zu jedem Element der Skalar addiert.

1.3.2 Multiplikation und Division

Multiplikation und Division werden mit * und / durchgeführt. Dies ist hierbei im Matrixsinne zu verstehen, d.h. * ist die Matrixmultiplikation und / ist die Multiplikation mit der Inversen ($A/B = A * B^{-1}$). Es ist dabei nur die Multiplikation einer $n \times m$ mit einer $m \times l$ Matrix möglich, wobei das Ergebnis eine $n \times l$ Matrix ist. Ausnahme ist hierbei wieder die Multiplikation mit einem Skalar. Jede Matrix kann mit einem Skalar multipliziert werden, wobei die Multiplikation dann elementweise durchgeführt wird.

1.3.3 Potenzieren

Potenzieren wird mit $\wedge$ durchgeführt. Dabei gilt das Gleiche, wie für die Multiplikation, d.h. Potenzieren ist im Matrixsinne zu verstehen und ist damit nur mit quadratischen Matrizen möglich.

1.3.4 Elementweise Operation

Alle vorigen Operationen können auch elementweise durchgeführt werden, indem man vor den Operator (+,-,*,/,$\wedge$) einen Punkt setzt (z.b. A.*B). Dabei wird dann die Operation skalar zwischen A(i,j) und B(i,j) durchgeführt, d.h.

$$C = A. * B \Rightarrow C_{ij} = A_{ij} \cdot B_{ij}$$

Dies ist natürlich nur für Matrizen gleicher Größe möglich.

1.3.5 Vordefinierte Funktionen

Matlab besitzt eine Vielzahl von vordefinierten Funktionen. Hier eine Aufzählung der wichtigsten:

exp(Matrix): Wendet elementweise die Exponentialfunktion auf die Matrix an

Trigonometrie: sin(Matrix), cos(), tan(), für die jeweiligen elementweisen Funktionen (im Bogenmaß), asin(), acos(), atan() für die entsprechenden Umkehrfunktionen.

expm(Matrix): Wendet die Matrixexponentialfunktion auf die Matrix an, d.h. die Matrix wird in die Exponentialreihe eingesetzt.

log(Matrix): Elementweiser natürlicher Logarithmus.

log2 oder log10(Matrix) Logarithmus zur Basis 2 bzw. 10.

sqrt(Matrix): Elementweise Wurzelfunktion.

': Transponieren einer Matrix, also $A' \Rightarrow A^T$

size: Gibt die Größe der Matrix an

1.4 Logische Operationen

Logische Operationen werden benötigt um Schleifen mit einer Abbruchbedingung anzuhalten, Fallunterscheidungen durchzuführen oder Werte in einer Matrix zu finden bzw. zu vergleichen, wie z.B. beim Sortieren. Selbst bei einfachen Skripten benötigt man also ziemlich schnell logische Operation. Als Ausgabewert einer logischen Operation erhält man stets entweder 0 für falsch oder 1 für wahr. Diese kann man dann verwenden um z.B. mit `if` etwas an-und auszuschalten oder mit den Werten einfach als Zahlen weiterzurechnen. Die folgenden Operationen können nun verwendet werden um Skalare zu vergleichen, woraufhin man als Antwort einen Skalar erhält. Vergleicht man Matrizen führt Matlab dies elementweise durch und gibt eine Matrix der gleichen Größe aus.

== : Vergleicht ob zwei Zahlen exakt gleich sind und gibt, wenn ja, 1 aus. Vorsicht, vermeintlich gleiche Zahlen können sich durch kleine numerische Fehler unterscheiden, wodurch 0 ausgegeben wird. Man sollte dann besser den Betrag der Differenz kleiner als eine Toleranz setzen: (abs(x1-x2)<tol).

>,<: Größer als, kleiner als

>=,<=: Größer gleich, kleiner gleich

&&: Logisches und. Steht zwischen zwei logischen Operationen und gibt wahr aus, wenn beide wahr sind.

||: Logisches oder. Steht ebenfalls ziwschen zwei logischen Operation, gibt aber wahr aus, wenn mindestens eine von beiden wahr ist.

∼: Negiert eine Aussage

1.5 Kontrollstrukturen

Zur Steuerung eines Programms werden Kontrollstrukturen verwendet. Dabei gibt es entweder Verzweigungen, meist durch `if` erzeugt, oder Schleifen, wobei hierfür meist `for` zum Einsatz kommt.

1.5.1 Schleifen mit for

Eine Schleife besitzt einen Anweisungsblock, welcher immer wieder durchgeführt wird, bis der Schleifenindex (im Bsp. j) durchgelaufen ist, oder sie von einer anderen Anweisung beendet wird. Der Anweisungsblock wird durch ein end abgegrenzt. Alles was zwischen `for (laufindex)` und `end` steht ist also der Anweisungsblock. Dabei kann auch eine weitere Schleife in diesem Anweisungsblock stehen. Beispiel:

```
1 n=0;
2 for j=1:50
3   n = n + 1;
4 end
```

Bei diesem Beispiel durchläuft die Schleife die Zahlen von 1 bis 50 und addiert bei jedem Durchlauf eins zu n, wodurch n am Ende den Wert 50 hat. Ein etwas realistischeres Beispiel die näherungsweise Berechnung von $\sum_{j=1}^{\infty} \frac{1}{j^2} = \frac{\pi^2}{6}$.

```
1 sum=0;
2 N = 10000;
3 for j=1:N
4   sum = sum + 1/j^2;
5 end
```

Hierbei wird zunächst der zu berechnende Wert mit 0 initialisiert. Dann wird in einer Schleife, die die natürlichen Zahlen bis N durchläuft, $\frac{1}{j^2}$ zu dem Wert addiert.

1.5.2 Verzweigungen mit if

Mit if wird eine Verzweigung erzeugt, d.h. if prüft eine dem Befehl folgende logische Aussage. Genau dann, wenn sie wahr ist, werden die Befehle im Anweisungsblock ausgeführt. Ist sie falsch und es steht kein `else` oder `elseif` im Anweisungsblock, springt das Programm zur Zeile nach dem `end`. Was hinter `else` steht wird ausgeführt, wenn die Aussage hinter `if` falsch ist. Bei `elseif` wird zusätzlich die logische Aussage hinter dem `elseif` geprüft. Wie bei der Schleife muss also der Anweisungsblock durch ein end abgegrenzt werden. Am einfachsten ist dies wahrscheinlich an einem Beispiel zu erklären:

```
1 n=0;
2 for j=1:50
3   if j<=25
4     n = n + 1;
5   else
6     n = n + 2;
7   end
8 end
```

Hierbei wird zunächst, wie im ersten Beispiel n initialisiert und eine Schleife begonnen welche j von 1 bis 50 hochzählt. Im Anweisungsblock wird allerdings nicht einfach $n = n + 1$ ausgeführt, sondern erst überprüft, ob $j \leq 25$ ist. Ist die Aussage wahr, wird der Anweisungsblock im `if` statement ausgeführt. Somit addiert das Skript bis $j = 26$ gilt eins zu n. Sobald $j > 25$ wird der Block hinter `else` ausgeführt und in jedem Schleifendurchgang 2 zu n addiert. Somit erhält man am Ende $n = 75$

1.6 Funktionen

Wir haben bereits eine Möglichkeit kennengelernt, selber Funktionen zu definieren (siehe Kapitel 1.2.3). Allerdings ist diese Methode so rudimentär wie simpel. Was aber machen wir, wenn wir komplexere Strukturen zu einer Einheit zusammenfassen möchten, sodass es eine definierte Ein- und Ausgabe gibt? Die Antwort ist: Indem wir als erste (echte) Programmzeile eines `m`-files den Header

```
1 function [out1, out2, ...] = funcname(in1, in2, ...)
```

schreiben, machen wir daraus eine Funktion, die von anderen Skripten aufgerufen werden kann. `funcname` wird hier durch den passenden Funktionsnamen ersetzt und die Variablen `in1`, `in2` etc. durch die Eingabevariablen und `out1`, `out2` etc. durch die Ausgabevariablen. Man muss dabei *nicht* die gleichen Variablennamen benutzen wie in dem Skript durch das man die Funktion aufrufen will. Allerdings muss der Name `funcname` der gleiche sein, wie der Name des `m`-files, unter den die Funktion abgespeichert wird. Beispiel:

```
1 function [mu, sigma] = StatEstimator(x)
2 n = length(x);
3 mu = sum(x)/n;
4 sigma = sqrt(sum(x-mu).^2/(n-1));
```

Das Beispiel zeigt, wie eine Funktion benutzt werden kann, um aus einem Datensatz automatisiert Mittelwert (`mu`) und Standardabweichung (`sigma`) auszulesen.

Alles, was auf den Header folgt, wird *body* genannt. Variablen, die im *body* definiert werden, sind nur lokal, d.h. man kann außerhalb der Funktion nicht auf sie zugreifen. Das hat den Vorteil, dass Variablen der Funktion und des aufrufenden Skriptes sich nicht gegenseitig überschreiben.

1.7 Plots

Abbildungen erzeugt man am einfachsten mit dem Befehl `plot`. Beispiel:

```
1 clear;
2
3 x = -2:0.1:2;
4 y = x.^2;
5
6 plot(x,y);
```

Das erzeugt ein Fenster wie links in Abb. 1.7.1. Erweitert man das obige Skript mit

```
7     xlabel('x');
8     ylabel('y');
9     title('Parabel');
```

so fügt man noch Achsenbeschriftungen und Titel hinzu.

Man kann das Aussehen des Plot ändern, indem man einen *linestyle* definiert, z.B. so:

```
6 plot(x,y,'ko');
```

Das `k` bedeutet dabei, dass mit schwarz (engl. *key*) gezeichnet wird. `o` legt die Marker fest. Da keine Linienart angegeben wurde, gibt es keine verbindende Linie zwischen

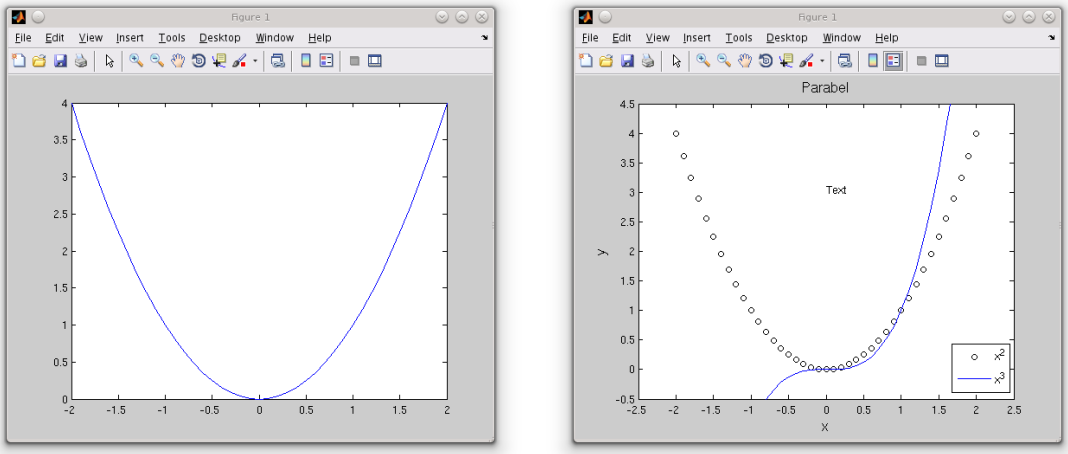

Abbildung 1.7.1: Links: Einfacher Plot einer Parabel. Rechts: Plot mit Achsenbeschriftungen, Titel, Legende und Textfeld.

des Datenpunkten. Eine Linienart gibt man z.B. so an: `'ko-'` oder `'k-'`, wenn man keine Marker haben möchte. Es gibt z.B. noch folgende weitere Optionen:

Farbe	**Marker**	**Linienart**
k(ey)	o	-
b(lue)	x	--
g(reen)	+	-.
r(ed)	s(quare)	:
y(ellow)	d(iamond)	
c(yan)		
m(agenta)		

Will man mehrere Plots in einer Grafik abbilden, so muss man den ersten Plot mit `hold` festhalten:

```
1  clear;
2
3  x = -2:0.1:2;
4  y = x.^2;
5
6  plot(x,y,'ko');
7  hold on
8     plot(x,x.^3)
9  hold off
```

Denn normalerweise wird bei jedem `plot`-Befehl das Grafikfenster neu initialisiert und der alte Plot gelöscht. Mit `hold on` verhindert man dies.

Man kann jetzt noch die Achsengrenzen festlegen, dem Plot eine Legende hinzufügen und einen Text irgendwo hinschreiben:

```
11     axis([-2.5 2.5 -0.5 4.5])
12     legend('x^2','x^3','Location','SouthEast');
13     text(0,3,'Text')
```

Der Befehl `axis` nimmt den Input `[Xmin Xmax Ymin Ymax]` und legt damit den sichtbaren Bereich des Plots fest. `legend` erzeugt eine Legende und platziert

sie da, wo die Option nach `'Location'` hinzeigt und der Befehl `text` erzeugt einfach ein Textfeld, hier an der Position $(x, y) = (0, 3)$ mit dem angegeben Text. Die Position wird in den tatsächlichen Koordinaten der Achsen angegeben.

Der vollständige Plot ist rechts in Abb. 1.7.1 zu sehen.

2 Theoretische Grundlagen

Im Folgenden sollen einige Grundlagen des Fittens besprochen werden. Obwohl wir uns sehr um Exaktheit bemühen, kann das vorliegende Skript keinen Anspruch auf Vollständigkeit erheben. Es soll vielmehr eine kurze und möglichst übersichtliche Einleitung in die Grundideen der Regression enthalten. Zu einer dieser Grundideen bzw. -annahmen zählt, dass alle Messgrößen gaußverteilt sind. Was das bedeutet, wird in den ersten Kapiteln erläutert. Dazu ist es auch notwendig den Unterschied zwischen Wahrscheinlichkeiten und Wahrscheinlichkeitsdichten zu verstehen. Die zweite Grundidee betrifft das Schätzen von wahren Größen aus den Messwerten. Will man eine direkt gemessene Größe aus mehreren Messwerten schätzen, so bedient man sich des Mittelwertes usw. Man kann aber eine physikalische Größe auch indirekt bestimmen, indem man sie als Fitparameter zweier direkt gemessener Größen berechnet. Dazu führt man eine neue Funktion der Messwerte und Parameter ein, die angeben soll, wie gut die Schätzung ist: das χ^2. Diese Methode wird *Methode der kleinsten Quadrate* genannt und ist die, die im Grundpraktikum angewendet wird. Wir werden versuchen zu erklären, warum das so ist.

Hat man die optimalen Fitparameter gefunden, bestimmt man ihre Unsicherheiten mit Hilfe der Kovarianzmatrix. Diese enthält auch Informationen darüber, wie die Fitparameter untereinander korreliert sind. Wir versuchen zu erklären, was das bedeutet.

2.1 Wahrscheinlichkeiten, Dichten und Schätzer

Wahrscheinlichkeiten. Die meisten werden mit dem Umgang mit Wahrscheinlichkeiten vertraut sein. Wenn wir z.B. einen perfekten sechseitigen Spielwürfel rollen, ist die Wahrscheinlichkeit, dass er am Ende eine 6 anzeigt, genau 1/6. Das gleiche gilt jeweils für die Augenzahlen 1 bis 5. Wenn X die Zufallsvariable ist, die angibt, welche Augenzahl gewürfelt wird, dann schreiben wir:

$$P(X = 6) = \frac{1}{6}.$$

Der **Erwartungswert** $\mu = \mathbb{E}(X)$ für X ist:

$$\mu = \mathbb{E}(X) = \frac{1 + 2 + 3 + 4 + 5 + 6}{6} = 3{,}5.$$

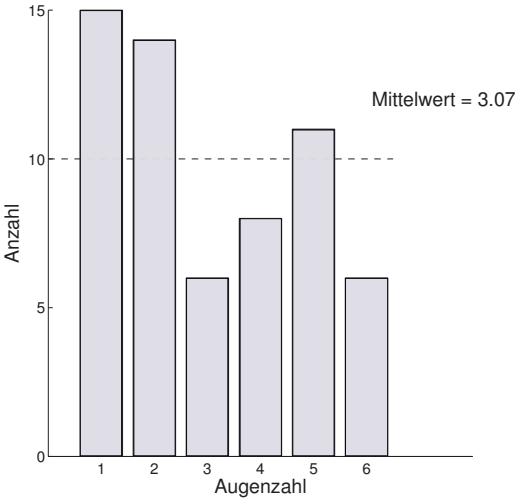

Abbildung 2.1.1: Würfelexperiment mit $N = 60$ Würfen. Der Mittelwert ist 3,07 während der Erwartungswert für die Augenzahl eines einzelnen Wurfes 3,5 ist.

Allgemein ist er für diskrete Zufallsvariablen
definiert als:

$$\mathbb{E}(X) = \sum_{k=-\infty}^{\infty} kP\left(X = k\right).$$

Ein Erwartungswert ist *kein* Mittelwert! Ein **Mittelwert** ergibt sich (in der Sprache des Würfelspiels), wenn man eine bestimmte Anzahl n von Würfen gewürfelt hat mit den Augenzahlen $X_1, \ldots X_n$ und dann den mittleren Wert

$$\overline{X}_{(n)} = \frac{X_1 + \ldots + X_n}{n}$$

berechnet. Somit kann man den Mittelwert als eine neue Zufallsvariable auffassen. Der Erwartungswert hingegen ist eine Eigenschaft der Zufallsvariable X an sich. In Abb. 2.1.1 ist der Ausgang eines solchen Würfelexperiment mit $n = 60$ gezeigt. Der Erwartungswert eines einzelnen Wurfes ist hier immer 3,5, während der Mittelwert in diesem besonderen Fall 3,07 ist. Erwartungswert und Mittelwert sind also nicht das gleiche.

Es gibt jedoch ein stochastisches Gesetz, dass besagt, dass unter besonderen Annahmen für jedes $\varepsilon > 0$ gilt:

$$\lim_{n\to\infty} P\left(|\overline{X}_{(n)} - \mu| > \varepsilon\right) = 0.$$

Dies wird in der Mathematik das **schwache Gesetz der großen Zahlen** genannt.

Es gibt noch einen anderen Zusammenhang zwischen Mittelwert und Erwartungswert:
Der Mittelwert $\hat{\mu} = \overline{X}$ einer Zufallsvariable X ist ein **erwartungstreuer Schätzer** für ihren Erwartungswert $\mu = \mathbb{E}(X)$. Das bedeutet, dass wenn wir den Erwartungswert μ nicht kennen, dann können wir ihn durch den Mittelwert $\hat{\mu}$ abschätzen und es gilt

$$\mathbb{E}(\hat{\mu}) \;=\; \mathbb{E}\left(\frac{X_1 + X_2 + \ldots X_n}{n}\right) \;=\; \frac{\mathbb{E}(X_1) + \mathbb{E}(X_2) + \ldots \mathbb{E}(x_n)}{n} \;=\; \frac{n\mu}{n} \;=\; \mu.$$

Man nennt eine solche Art von Schätzern auch *unbiased*, in dem Sinne, dass sie die wahre Größe gleichermaßen über- oder unterschätzen, aber nicht zu einer Seite tendieren.

Weiterhin kennen wir außer dem Erwartungswert noch (mindestens) eine andere wichtige Größe zur Charakterisierung von Zufallsvariablen, nämlich die Varianz $\sigma^2 = \mathbb{E}(X^2) - (\mathbb{E}X)^2$. Für diese Größe hätten wir auch gerne einen erwartungstreuen Schätzer $\hat{\sigma}^2$ und wir machen den Ansatz:

$$\hat{\sigma}^2 = \frac{\sum_{i=1}^{n}(X_i - \overline{X})^2}{C},$$

mit einer noch zu bestimmenden Konstante C. Der wichtige Punkt ist nun, dass wir in der Summe nicht μ, sondern $\overline{X} = \hat{\mu}$ stehen haben. Dieses $\overline{X}$ ist wie bereits gesagt wiederum eine Zufallsvariable. Was ist der Erwartungswert von $\hat{\sigma}^2$?

$$\mathbb{E}(\hat{\sigma}^2) \;=\; \mathbb{E}\left(\frac{\sum_{i=1}^{n}(X_i - \overline{X})^2}{C}\right) \;=\; \frac{\sum_{i=1}^{n}\mathbb{E}\left((X_i - \overline{X})^2\right)}{C}.$$

Man darf das Quadrat niemals aus dem Erwartungswert herausziehen. Stattdessen muss man ausmultiplizieren:

$$\begin{aligned}\mathbb{E}(\hat{\sigma}^2) &= \frac{\sum_{i=1}^n \mathbb{E}\left(X_i^2 - 2X_i\overline{X} + \overline{X}^2\right)}{C} \\ &= \frac{\sum_{i=1}^n \left(\mathbb{E}(X_i^2) - 2\mathbb{E}(X_i\overline{X}) + \mathbb{E}(\overline{X}^2)\right)}{C} \\ &= \frac{1}{C}\sum_{i=1}^n \left[\mathbb{E}(X_i^2) - 2\mathbb{E}\left(\frac{X_i\sum_k X_k}{n}\right) + \mathbb{E}\left(\frac{(\sum_k X_k)^2}{n^2}\right)\right]\end{aligned}$$

Wenn $i \neq k$, dann faktorisiert der Erwartungswert im mittleren Term:

$$\begin{aligned}\mathbb{E}(\hat{\sigma}^2) &= \frac{1}{C}\sum_{i=1}^n \left[\mathbb{E}(X_i^2) - \frac{2}{n}\mathbb{E}(X_i^2) - 2\mathbb{E}\left(\frac{X_i\sum_{k\neq i} X_k}{n}\right)\right. \\ &\qquad \left. + \frac{1}{n^2}\,\mathbb{E}\left(\sum_k X_k^2 + \sum_k\sum_{k\neq m} X_k X_m\right)\right]\end{aligned}$$

Ebenso faktorisiert der Ewartungswert, wenn im letzten Term $k \neq m$:

$$\begin{aligned}\mathbb{E}(\hat{\sigma}^2) &= \frac{1}{C}\sum_{i=1}^n \left[\frac{n-2}{n}\,\mathbb{E}(X_i^2) - \frac{2(n-1)}{n}\,\mu^2 + \frac{1}{n}\mathbb{E}(X_i^2) + \frac{1}{n^2}\,\mathbb{E}\left(\sum_k\sum_{k\neq m} X_k X_m\right)\right] \\ &= \frac{1}{C}\sum_{i=1}^n \left[\frac{n-1}{n}\,\mathbb{E}(X_i^2) - \frac{2(n-1)}{n}\,\mu^2 + \frac{n(n-1)}{n^2}\,\mu^2\right] \\ &= \frac{1}{C}\sum_{i=1}^n \left[\frac{n-1}{n}\,\mathbb{E}(X_i^2) - \frac{n-1}{n}\,\mu^2\right] = \frac{n-1}{nC}\sum_{i=1}^n \left[\mathbb{E}(X_i^2) - \mu^2\right]\end{aligned}$$

Jetzt muss man bedenken, dass die X_i identisch verteilt sind, also $\mathbb{E}(X_i^2) = \mathbb{E}(X_1^2)$ für alle i:

$$\mathbb{E}(\hat{\sigma}^2) = \frac{n-1}{C}\,\left[\mathbb{E}(X_1^2) - \mu^2\right] = \frac{n-1}{C}\,\sigma^2.$$

Da $\mathbb{E}(\hat{\sigma}^2)$ gleich σ^2 sein soll, ist $C = n-1$.

Wahrscheinlichkeitsdichten. Das Beispiel mit den Würfeln veranschaulicht, was Wahrscheinlichkeiten sind. Diese Wahrscheinlichkeiten ordnet man bestimmten Ereignissen zu, wie z.B. eine 6 zu würfeln. Für das folgende ist es aber auch wichtig zu verstehen, was Wahrscheinlichkeitsdichten sind. Dazu untersuchen wir ein anderes Beispiel: Stellen wir uns vor, wir hätten ein sehr großes Stück Papier und ein sehr kleines Lineal, sodass wir das Lineal mehrmals neu ansetzen müssen, um das Papier auszumessen. Weil wir wissen, wie ungenau diese Messmethode ist, wiederholen wir

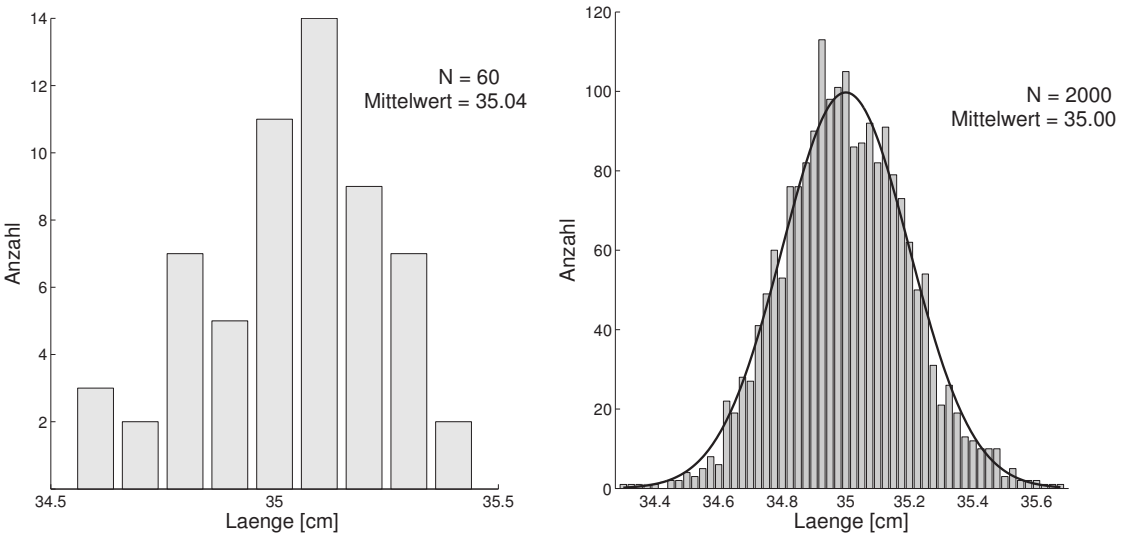

Abbildung 2.1.2: Vermessung der Länge eines Papierblattes mit einem Lineal. Die Messung wird jeweils N mal durchgeführt und notiert, wie oft ein bestimmter Wert gemessen wird. Die Rasterung ist 1 mm im linken und 0,25 mm im rechten Bild. Die Kurve rechts zeigt eine entsprechend normierte Gaußkurve.

die Messung z.B. 60 mal. Das Ergebnis dieser $N = 60$ Messungen ist links in Abb. 2.1.2 dargestellt.

Wenn wir nun stattdessen 2000 mal messen und die Rasterung der Messwerte genauer machen, dann erhalten wir z.B. das Ergebnis rechts in Abb. 2.1.2. Die dazu eingezeichnete Kurve entspricht einer (passend normierten) Gaußkurve. Mit ordentlicher Normierung lautet die Definition einer **Gaußverteilung**:

$$\mathcal{N}_{\mu,\sigma}(x) = \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \tag{2.1.1}$$

mit Erwartungswert μ und Standardabweichung σ. Man nennt sie auch **Normalverteilung**. *Ordentliche Normierung* heißt, dass

$$\int_{-\infty}^{\infty} \mathcal{N}_{\mu,\sigma}(x)dx = 1. \tag{2.1.2}$$

Diese Normalverteilung ist ein Beispiel für eine Wahrscheinlichkeitsdichte, d.h. sie gibt keine Wahrscheinlichkeiten direkt an. An die Wahrscheinlichkeiten kommt man erst per Integration. Wenn man eine normalverteilte Zufallsvariable X mit Erwartungswert μ und Standardabweichung σ hat, ergibt es nicht viel Sinn zu fragen, was die Wahrscheinlichkeit ist, dass X genau π ist. Sie ist Null. Allerdings kann man fragen, was die Wahrscheinlichkeit ist, dass X in einem bestimmen Intervall $[\pi - \varepsilon, \pi + \varepsilon]$ mit $\varepsilon > 0$ liegt. Dann erhält man folgende Anwort:

$$P(X \in [\pi - \varepsilon, \pi + \varepsilon]) = \int_{\pi-\varepsilon}^{\pi+\varepsilon} \mathcal{N}_{\mu,\sigma}(x)dx \neq 0. \tag{2.1.3}$$

Allgemein ist die Wahrscheinlichkeit, dass die Zufallsvariable X in den Grenzen $[a, b]$ liegt:

$$P(X \in [a,b]) = \int_a^b \mathcal{N}_{\mu,\sigma}(x)dx. \tag{2.1.4}$$

Deshalb muss die Normierung auch wie in Gl. (2.1.2) sein, denn:

$$1 = P(X \in \mathbb{R}) = \int_{-\infty}^{\infty} \mathcal{N}_{\mu,\sigma}(x)dx. \tag{2.1.5}$$

Man überzeugt sich, dass μ tatsächlich der Erwartungswert ist:

$$\mu = \int_{-\infty}^{\infty} x\mathcal{N}_{\mu,\sigma}(x)dx,$$

und für die Varianz σ^2 gilt:

$$\sigma^2 = \int_{-\infty}^{\infty} (x-\mu)^2\mathcal{N}_{\mu,\sigma}(x)dx.$$

Die Varianz σ^2 bestimmt die Breite der Verteilung und spielt in der Statistik die Rolle einer Unsicherheit. Die Wahrscheinlichkeit, einen Wert innerhalb der **1σ-Umgebung** zu ziehen, ist

$$P\left(\mu - 1\sigma \leq X \leq \mu + 1\sigma\right) = \int_{\mu-\sigma}^{\mu+\sigma} \mathcal{N}_{\mu,\sigma}(x)\ dx \approx 0{,}683. \tag{2.1.6}$$

Dies nennt man auch die *coverage probability* (auch statistische Sicherheit) der 1σ-Umgebung. In Formelzeichen wird sie auch $1-\alpha$ genannt, während α die sogenannte Restwahrscheinlichkeit ist, also die Wahrscheinlichkeit, einen Wert außerhalb der 1σ-Umgebung zu ziehen. Wählt man nicht σ als Unsicherheit sondern einen größeren Wert wie z.B. 2σ, dann vergrößert sich auch die statistische Sicherheit.

Es gibt natürlich unendlich viele andere Wahrscheinlichkeitsdichten bzw. Verteilungen. Bekannte Vertreter sind z.B.

- die Exponentialverteilung:

$$f_\lambda(x) = \begin{cases} \lambda e^{-\lambda x}, & x \geq 0, \\ 0, & \text{sonst.} \end{cases}$$

  und

- die Pareto-Verteilung:

$$f_{\alpha,x_0} = \begin{cases} \frac{\alpha x_0^\alpha}{x^{1+\alpha}}, & x \geq x_0 \\ 0, & \text{sonst.} \end{cases}$$

3 Theorie: Nichtlineare Regression

3.1 Grundannahme und Zentraler Grenzwertsatz

Im zweiten Beispiel (Abb. 2.1.2) des vorherigen Kapitels hatten wir gesehen, dass die Messergebnisse normalverteilt um einen bestimmten empirischen Mittelwert waren. Das war kein Zufall, denn es handelte sich nicht um ein reales Experiment, sondern um eine Simulation, die genauso programmiert war. Allerdings war auch das nicht willkürlich, denn die Simulation folgt somit der wichtigsten Annahme des Fittens:

Grundannahme

Alle Messgrößen sind normalverteilt.

Diese Annahme selber hat ihre Wurzeln im **Zentralen Grenzwertsatz**. Eine Version dieses Satzes lautet:

> Seien $X_1, X_2, \ldots$ eine Folge von unabhängigen Zufallsvariablen mit endlichen Erwartungswerten μ_i und Varianzen σ_i^2. Sei $s_n^2 = \sum_{i=1}^n \sigma_i^2$ und sei die Feller-Bedingung erfüllt. Dann konvergiert
>
> $$\sum_{i=1}^{n} X_i \quad \text{für} \quad n \to \infty \quad \text{in Verteilung gegen} \quad \mathcal{N}\left(\sum_{i=1}^{n} \mu_i,\ \sqrt{s_n^2}\right).$$

Was die Feller-Bedingung ist, interessiert uns jetzt nicht. Wir nehmen einfach an, dass sie erfüllt ist. In Worten ausgedrückt bedeutet der Grenzwertsatz, dass die Summe vieler unabhängiger Zufallsvariablen mit endlichen Erwartungswerten und Varianzen *ungefähr* normalverteilt ist mit Erwartungswert $\sum \mu_i$ und Varianz $\sum_i \sigma_i^2$.

Man sagt nun, dass eine Messung von vielen kleinen unabhängigen Zufallsvariablen X_i beeinflusst wird (vgl. auch zufällige Fehler), die sich so addieren, dass der Messwert als normalverteit angenommen werden kann. Dies rechtfertigt die obige Grundannahme.

3.2 Maximum Likelihood und Kleinste-Quadrate-Methode

Es wird nun Zeit, sich mehr dem Fitten von Daten zuzuwenden.

Angenommen, man hat N (unabhängige) Messwerte y_i mit Unsicherheiten σ_i zu den Grundvariablen x_i, die wir als sicher annehmen. Weiterhin gibt es eine Funktion

$f(.|\boldsymbol{\theta}) : \mathbb{R} \to \mathbb{R}$, die von dem Parameterset $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_m)$ abhängt. Die Hypothese sagt, dass der funktionelle Zusammenhang zwischen den x_i und den y_i die Form von f hat und die Aufgabe ist, die Parameter $\boldsymbol{\theta}$ so zu bestimmen, dass die $f(x_i|\boldsymbol{\theta})$ *besonders gut* zu den y_i passen.

Was bedeutet besonders gut? Eine mögliche Antwort gibt die **Maximum Likelihood-Methode**: Sie sagt, die Wahl der Paramater θ ist dann besonders gut, wenn die zusammengesetzte Wahrscheinlichkeitsdichte

$$\varphi(\mathbf{z}|\boldsymbol{\theta}) = \prod_{i=1}^{N} \mathcal{N}_{f(x_i|\boldsymbol{\theta}),\sigma_i}(z_i) \tag{3.2.1}$$

an der Stelle $\mathbf{z} = (y_1, \ldots y_N)$ maximal ist. Diese zusammengesetze Wahrscheinlichkeitsdichte ist das kontinuierliche Analogon zu der Wahrscheinlichkeit, das mehrere unabhängige Ereignisse gleichzeitig eintreten. Seien z.B. $A_1, \ldots, A_n$ unabhängige Ereignisse, die entweder eintreten oder nicht. Dann ist die Wahrscheinlichkeit, dass alle Ereignisse gleichzeitig eintreten:

$$P\left(A_1 \cap \ldots \cap \ A_n\right) = \prod_{i=1}^{n} P(A_i).$$

Wenn die Dichte $\varphi(.|\boldsymbol{\theta})$ in Gl. (3.2.1) aber maximal ist, dann (und genau dann) ist auch ihr Logarithmus maximal, denn der Logarithmus ist eine streng monoton steigende Funktion. Man soll also folgende Funktion bezüglich $\boldsymbol{\theta}$ maximieren:

$$\begin{aligned}
\ln \varphi(\mathbf{y}|\boldsymbol{\theta}) &= \ln \prod_{i=1}^{N} \mathcal{N}_{f(x_i|\boldsymbol{\theta}),\sigma_i}(y_i) = \sum_{i=1}^{N} \ln \mathcal{N}_{f(x_i|\boldsymbol{\theta}),\sigma_i}(y_i) \\
&= \sum_{i=1}^{N} \left[\ln \left(\frac{1}{\sqrt{2\pi\sigma_i^2}} \right) - \frac{(y_i - f(x_i|\boldsymbol{\theta}))^2}{2\sigma_i^2} \right] \\
&= -\frac{1}{2} \left[\sum_{i=1}^{N} \ln(2\pi\sigma_i^2) + \sum_{i=1}^{N} \frac{(y_i - f(x_i|\boldsymbol{\theta}))^2}{\sigma_i^2} \right].
\end{aligned} \tag{3.2.2}$$

Die erste Summe wird in unserem Fall immer konstant bleiben, weil wir die σ_i als gegeben annehmen. Genauso spielt auch der Vorfaktor 1/2 bei der Maximierung keine Rolle. Es bleibt also noch, $\boldsymbol{\theta}$ so zu bestimmen, dass

$$-\sum_{i=1}^{N} \frac{(y_i - f(x_i|\boldsymbol{\theta}))^2}{\sigma_i^2} =: -\chi^2$$

maximal wird. Das ist aber das gleiche, als würden wir

$$\chi^2 = \sum_{i=1}^{N} \frac{(y_i - f(x_i|\boldsymbol{\theta}))^2}{\sigma_i^2} \tag{3.2.3}$$

minimieren. Somit führt uns die Maximum Likelihood-Methode in unserem Spezialfall direkt zur Methode der kleinsten Quadrate (*least squares*). Wir bestimmen unsere Parameter $\boldsymbol{\theta}$ also so, dass die χ^2-Funktion also die Summe der quadratischen Abweichungen relativ zu den Unsicherheiten σ_i minimal wird.

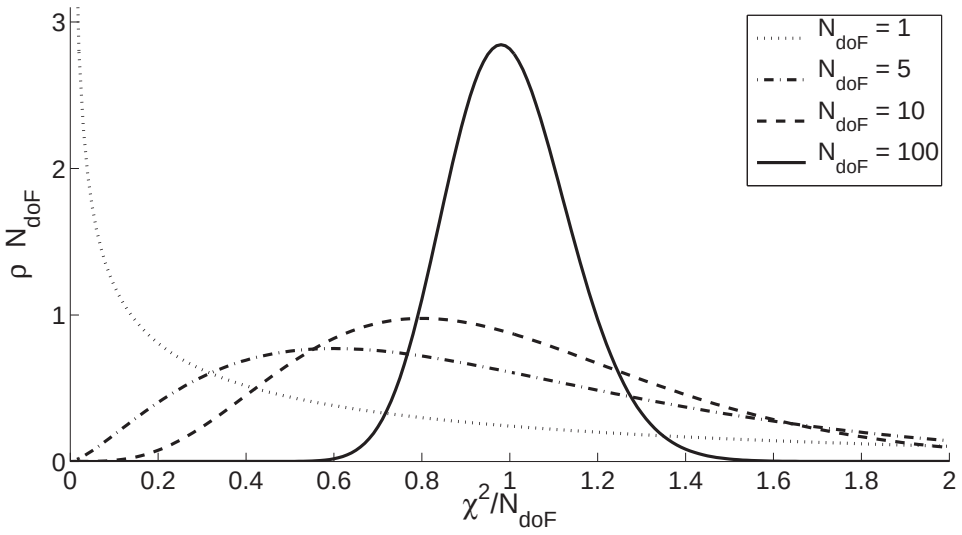

Abbildung 3.2.1: χ^2-Verteilung für verschiedene Anzahlen von Freiheitsgraden N_{doF}.

Hypothesentest. Nachdem man die Parameter $\boldsymbol{\theta}$ bestimmt hat, möchte man vielleicht noch eine Auskunft darüber erhalten, wie gut die Hypothese f überhaupt war. Vielleicht hat man ja fälschlicherweise einen linearen Zusammenhang zwischen den Größen x und y angenommen, wo gar keiner war. Dazu bestimmt man die Wahrscheinlichkeitsdichte $\rho(\chi^2)$, dass ein bestimmter Wert von χ^2 eintritt, gegeben, dass man eine bestimmte Anzahl N_{doF} von **Freiheitsgraden** (*degrees of freedom*) hat. Die Zahl der Freiheitsgrade ist die Zahl N der Messwerte minus die Zahl m der Fitparameter:

$$N_{\mathrm{doF}} = N - m.$$

Falls man sich für den Rechenweg interessiert, so findet man diesen im CPI-Skript [1] Kapitel 9. Abb. 3.2.1 zeigt eine grafische Darstellung dieser Wahrscheinlichkeitsdichte $\rho(\chi^2)$ nachnormiert nach der Zahl der Freiheitsgrade. Man sieht, dass für das sogenannte reduzierte χ^2, also χ^2/N_{doF} und sehr große Zahl von Freiheitsgraden N_{doF} die Wahrscheinlichkeitsdichte bei Eins maximal ist. Bei sehr viel größeren Werten von χ^2/N_{doF} ist die Hypothese zu verwerfen und bei sehr viel kleineren Werten hat man vermutlich seine Unsicherheiten zu groß abgeschätzt.

Eine Größe, die hierzu oft angegeben wird, ist der Q-Wert des Fits. Dieser ist nichts anderes als die Wahrscheinlichkeit

$$Q(z) = P(\chi^2 \geq z). \tag{3.2.4}$$

Nach dem Skript [1] kann man der Hypothese bis $Q = 0{,}1$ vertrauen.

Tabelle 3.2.1: $\Delta\chi^2$ für verschiedene Anzahl m der Parameter und Werte von $1-\alpha$. Tabelle aus [2].

$(1-\alpha)$ [%]	$m=1$	$m=2$	$m=3$
68,27	1,00	2,30	3,53
90,	2,71	4,61	6,25
95,	3,84	5,99	7,82

3.2.1 Unsicherheiten der Parameter

Nachdem man die optimalen Parameter $\hat{\boldsymbol{\theta}}$ geschätzt hat, stellt sich noch die Frage nach ihren Konfidenzintervallen. Angenommen, wir hätten zwei Parameter geschätzt: a und b. Dann erhält man die Unsicherheiten σ_a und σ_b daraus, dass man sich die Niveauline der $\chi^2(a,b)$-Funktion anschaut, wo der Wert $\chi^2_{\min}+\Delta\chi^2$ ist mit $\Delta\chi^2 = 2{,}30$. Im linearen Fall (siehe Kapitel 5) ist dies eine Ellipse wie in Abb. 3.2.2 schematisch dargestellt. Die Neigung ist der Ellipse ist auf eine Korrelation der Parameter a und b zurückzuführen, in diesem Fall eine negative Korrelation. Im Fall unkorrelierter Variablen sind die Hauptachsen parallel zu den Koordinatenachsen.

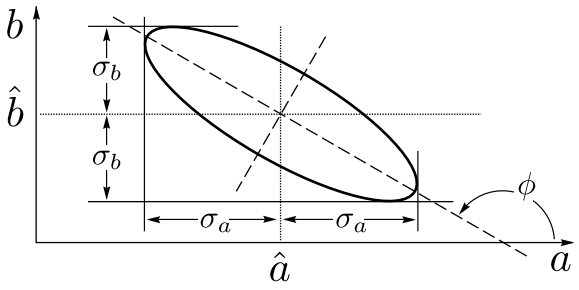

Abbildung 3.2.2: Fehlerellipse. Schematische Darstellung der Niveaulinie von $\chi^2_{\min}+\Delta\chi^2$ in den Variablen a und b im Fall negativer Korrelation. Im nichtlinearen Fall ist dies nicht notwendig eine Ellipse. Adaptiert von Fig. 5 Kapitel *Statistics* aus [2].

Je nachdem wie viele Parameter man schätzt, ändert sich auch der passende Wert für $\Delta\chi^2$. Das gleiche gilt selbstverständlich auch, wenn man nicht das 68%-Intervall, sondern ein anderes schätzen möchte. Tabelle 3.2.1 zeigt einige Werte von $\Delta\chi^2$ zu verschiedenen Werten von m und $1-\alpha$.

4 Numerik: Nichtlineare Regression

4.1 Praktische Implementierung einer nichtlinearen Regression anhand wnonlinfit

Ziel der nichtlinearen Regression ist es, eine Funktion $f(\mathbf{x}|\boldsymbol{\theta})$ mit den freien Parametern $\boldsymbol{\theta}$ an gemessene Wertetupel x_i und y_i mit den Unsicherheiten des y-Wertes σ_i anzupassen. In Kapitel 3.2 haben wir bereits gesehen, dass bei normalverteilten Fehlern es optimal ist

$$\chi^2(\boldsymbol{\theta}) = \sum_i \frac{(y_i - f(x_i|\boldsymbol{\theta}))^2}{\sigma_i^2}, \tag{4.1.1}$$

zu minimieren. Im nichtlinearen Fall, d.h. wenn $f(x|\boldsymbol{\theta})$ nicht linear in $\boldsymbol{\theta}$ ist, ist dies nicht im Allgemeinen analytisch möglich, wodurch wir auf numerische Routinen angwiesen sind.

4.1.1 Numerische Optimierung

Mathematisch gesehen kann man unser Problem darauf reduzieren, das globale Minimum einer skalarwertigen Funktion $g(\mathbf{x})$ zu bestimmen. Das $\mathbf{x}$ in der Funktion $g(\mathbf{x})$ darf nicht mit den x-Werten in der Messung verwechselt werden, sondern spielt hier die Rolle von $\boldsymbol{\theta}$, während g die Rolle von χ^2 einnimmt. Praktisch gesehen ist es allerdings nahezu unmöglich das globale Minimum zu finden, weswegen alle numerisch durchführbaren Routinen nur lokale Minima suchen und finden. Die Idee dieser numerischen Methoden ist es dabei bei einem meist durch den User vorgegeben Startwert anzufangen und sich dem Minimum dann iterativ anzunähern. Ob das Verfahren das globale Minimum findet oder überhaupt konvergiert hängt dabei kritisch von dem vorgegebenen Startwert ab. Es ist also nötig einen sinnvollen Startwert zu schätzen.

Newton-Verfahren. Wir wollen mit dem Newton-Verfahren, beginnend von einem Startpunkt $\mathbf{x}_0$, das Minimum $\mathbf{x}_{\min}$ der Funktion $g(\mathbf{x})$ bestimmen. Dazu approximieren wir die Funktion $g(\mathbf{x})$ zunächst durch ihre (mehrdimensionale) Taylorreihe am Startpunkt $\mathbf{x_0}$ bis zur 2. Ordnung. Die approximierte Funktionen nennen wir $\tilde{g}(\mathbf{x})$

$$\tilde{g}(\mathbf{x}) := g(\mathbf{x_0}) + \boldsymbol{\nabla} g(\mathbf{x})|_{\mathbf{x}_0} \cdot (\mathbf{x} - \mathbf{x_0}) + \frac{1}{2}(\mathbf{x} - \mathbf{x_0})^T \cdot H(g(\mathbf{x}))|_{\mathbf{x}_0} \cdot (\mathbf{x} - \mathbf{x_0}) \approx g(\mathbf{x}), \tag{4.1.2}$$

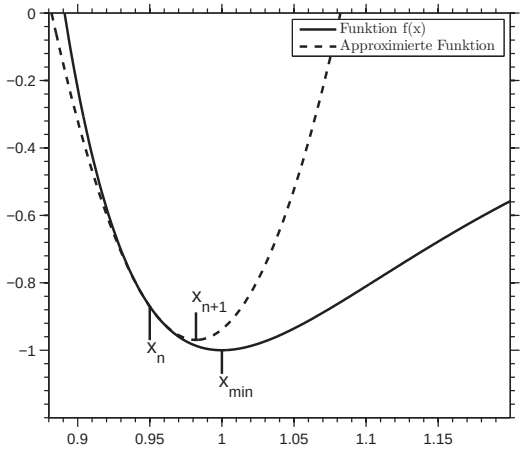

Abbildung 4.1.1: Skizze eines Iterationsschrittes des Newton-Verfahrens.

wobei $H(g(x))|_{\mathbf{x}_0}$ die Hesse-Matrix von $g(\mathbf{x})$ an der Stelle $\mathbf{x_0}$ ist und durch

$$H(g(\mathbf{x}))|_{\mathbf{x}_0} := \begin{pmatrix} \frac{\partial^2 g}{\partial x_1 \partial x_1}|_{\mathbf{x}_0} & \frac{\partial^2 g}{\partial x_1 \partial x_2}|_{\mathbf{x}_0} & \cdots & \frac{\partial^2 g}{\partial x_1 \partial x_n}|_{\mathbf{x}_0} \\ \frac{\partial^2 g}{\partial x_2 \partial x_1}|_{\mathbf{x}_0} & \frac{\partial^2 g}{\partial x_2 \partial x_2}|_{\mathbf{x}_0} & \cdots & \frac{\partial^2 g}{\partial x_2 \partial x_n}|_{\mathbf{x}_0} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 g}{\partial x_n \partial x_1}|_{\mathbf{x}_0} & \frac{\partial^2 g}{\partial x_n \partial x_2}|_{\mathbf{x}_0} & \cdots & \frac{\partial^2 g}{\partial x_n \partial x_n}|_{\mathbf{x}_0} \end{pmatrix}.$$

gegeben ist. Im Minimum der Funktion $\tilde{g}(\mathbf{x})$ muss natürlich der Gradient der Funktion verschwinden, d.h. es gilt

$$0 = \boldsymbol{\nabla} \tilde{g}(\mathbf{x})|_{x_{\min}} \approx \boldsymbol{\nabla} g(\mathbf{x})|_{\mathbf{x}_0} + H(g(\mathbf{x}))|_{\mathbf{x}_0} \cdot (\mathbf{x}_{\min} - \mathbf{x_0}) \tag{4.1.3}$$

Auflösen nach $x_{\min}$ führt auf

$$\mathbf{x}_{\min} = \mathbf{x_0} - (H(g(\mathbf{x}))|_{\mathbf{x}_0})^{-1} \cdot \boldsymbol{\nabla} g(\mathbf{x})|_{\mathbf{x}_0} \tag{4.1.4}$$

Da dies natürlich nur korrekt ist, wenn die Funktion wirklich quadratisch in jeder Variable ist führt man die Iteration

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \cdot (H(g(\mathbf{x}))|_{x_n})^{-1} \cdot \boldsymbol{\nabla} g(\mathbf{x})|_{x_n} \tag{4.1.5}$$

ein. Diese wiederholt man so lange bis eine geeignete Abbruchbedingung die Schleife beendet. γ_n ist dabei die Schrittweite. Sie sollte stets zwischen 1 und 0 liegen und wird meist zu Beginn kleiner gewählt um nicht möglicherweise zu große Schritte zu machen. Befindet man sich schon nahe dem Minimum kann sie auf 1 gesetzt werden um schnellere Konvergenz zu erreichen. Für $\gamma_n = 1$ konvergiert das Verfahren quadratisch, d.h. $|\mathbf{x}_n - \mathbf{x}_{\min}| = \mathcal{O}\left(\frac{1}{n^2}\right)$

Gradienten-Verfahren (steepest descent). Ein weiteres deutlich robusteres Verfahren ist das sogenannte Gradienten-Verfahren. Es basiert auf der Idee, stets der

Richtung des steilsten Abstieges zu folgen, d.h. entlang des negativen Gradientens. Die Iterationsformel lautet nun

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \cdot \boldsymbol{\nabla} g(\mathbf{x})|_{x_n} \tag{4.1.6}$$

Die Schrittweite wird dabei wieder durch einen Parameter γ_n festgelegt, der durch ein weiteres Verfahren festgelegt werden muss. Man kann beispielsweise eine 1-D Optimierung entlang der durch den Gradienten festgelegten Geraden durchführen, was im Allgemeinen sehr viel einfacher als eine mehrdimensionale Optimierung ist. Das Verfahren ist im allgemeinen sehr robust, da es garantiert, dass der Funktionswert gesenkt wird. Allerdings neigt es dazu, um das eigentlich Minimum in Zickzack-Schritten herumzulaufen. Deshalb ist die Konvergenz nahe dem Minimum auch eher schlecht.

Implementierung in wnonlinfit. In `wnonlinfit` haben wir die nichtlineare Regression mit der Matlab-Routine `lsqnonlin` implementiert. Diese verwendet das sogennante Trust-Region-Verfahren. Dieses basiert grundsätzlich auf dem Newton-Verfahren, allerdings wird dabei der quadratischen Approximation nur in einem gewissen Umfeld (deswegen „Trust-Region“) vertraut. Man definiert einen Trust-Region-Radius Δr, und beschränkt den Raum, in dem die Funktion $g(\mathbf{x})$ minimiert werden soll, auf z.B. eine Kugel mit diesem Radius. Deshalb kann man auch nicht einfach Gl. 4.1.5 verwenden, sondern muss eine sogenannte restringierte Optimierung durchführen. Restringierte Optimierung bedeutet dabei, dass das Minimum nicht in ganz $\mathbb{R}$ gesucht wird, sondern nur auf einer Teilmenge davon, z.B. eben innerhalb einer Kugel. Dies ist allerdings bei einer quadratischen Funktion analytisch mit Hilfe der Methode der Lagrange-Multiplikatoren möglich, welche einen Schrittvektor $\Delta\mathbf{x}$ als Ergebnis liefert. Daraufhin bildet man das Verhältnis Γ_k aus der tatsächlichen Verringerung der Funktion und der vorhergesagten Verringerung

$$\Gamma_n = \frac{g(\mathbf{x_n} + \Delta\mathbf{x}) - g(\mathbf{x_n})}{\tilde{g}(\mathbf{x_n} + \Delta\mathbf{x}) - g(\mathbf{x_n})}$$

Ist $\Gamma_n \geq \rho_1$ war das Verfahren erfolgreich und der Radius kann beibehalten oder vergrößert werden und der Schritt wird durchgeführt, d.h. $\mathbf{x}_{n+1} = \mathbf{x}_n + \Delta\mathbf{n}$. Dabei ist ρ_1 ein vordefinierter Parameter. Je nach Wahl agiert die Routine riskanter und schneller (bei $\rho_1 \ll 1$) oder vorsichtiger und langsamer (bei $\rho_1 \lesssim 1$). Ist hingegen $\Gamma_n < \rho_1$ gilt der Schritt als nicht erfolgreich, d.h. $\mathbf{x}_{n+1} = \mathbf{x}_n$ und der Trust-Region-Radius wird verkleinert.

4.2 Nichtlinearer Fit mit wnonlinfit

Die Syntax für einen Aufruf von wlinfit lautet

$$\left[\boldsymbol{\theta},\ \boldsymbol{\theta}_{\mathrm{err}},\ U_{\mathrm{kov}},\ \chi^2/\mathrm{DoF}\right] = \mathrm{wnonlinfit}\left(\mathbf{x}, \mathbf{y}, \mathbf{y}_{\mathrm{err}}, \mathrm{fitfunc}, \boldsymbol{\theta}_0\right) \tag{4.2.1}$$

Die Eingabeparameter sind dabei:

$\mathbf{x}$: Vektor der X-Werte

$\mathbf{y}$: Vektor der Y-Werte

$\mathbf{y}_{\text{err}}$: Vektor der Unsicherheiten $\sigma_{\mathbf{y}}$ der Y-Werte

fitfunc: Function handle mit der zu fittenden Funktion. Die Form ist dabei $f(x, c)$, die Funktion muss somit zwei Eingabevariablen besitzen. Die erste ist ein Vektor der x-Werte, die zweite ein Vektor der Fitparameter.

$\boldsymbol{\theta}_0$: Anfangsschätzung für die Fitparameter $\boldsymbol{\theta}$. Hierbei muss beachtet werden, dass wenn die Schätzung zu schlecht ist der Fit unter Umständen nicht konvergiert.

Die Ausgabewerte sind:

$\boldsymbol{\theta}$: Vektor der Koeffizienten des Fits, also die Ergebnisse.

$\boldsymbol{\theta}_{\text{err}}$: Vektor der Unsicherheiten von $\boldsymbol{\theta}$

U_{kov}: Kovarianzmatrix

χ^2/doF: Der χ^2-Wert geteilt durch die Anzahl der Freiheitsgrade. Dieser sollte nahe bei 1 liegen.

Zusätzlich werden wie zwei Plots erstellt. Der erste enthält den Fit mit den Ergebnissen in einer Legende, der andere die Residuen. Da `wnonlinfit` numerische Optimierungsroutinen verwendet (also Routinen, die das globale Minimum suchen), braucht es Startwerte, die vorgeben, wo die Suche nach dem Minimum beginnen soll. Dabei kann es zu zwei grundlegenden Problemen bei der Minimierung kommen: Der erste problematische Fall ist, dass nicht das globale, sondern nur ein lokales Minimum von $\frac{\chi^2}{\text{dof}}$ gefunden wurde. Dies äußert sich meistens in einem recht hohen $\frac{\chi^2}{\text{dof}}$. Dies passiert beispielsweise bei Fits mit einer Gaussfunktion, wenn der geschätzte Mittelwert zu stark vom wahren Wert abweicht. Typischerweise endet dann der Fit in einem Minimum, bei dem die Fitfunktion einfach identisch null ist. Der zweite Fall, der eintreten kann, ist, dass die Routine gar nicht konvergiert und irgendwann einfach abbricht und den letzten Wert, den sie berechnet hat, ausgibt. Um die Wahrscheinlichkeit eines erfolgreichen Fits trotz schlechter Startwerte zu erhöhen, führt `wnonlinfit`, wenn $\frac{\chi^2}{\text{dof}}$ zu schlecht war, eine zufällige Variation der vorgegebenen Startwerte durch. Dies wird als Iteration angezeigt und erfolgt so lange, bis die gewünschte Toleranz in $\frac{\chi^2}{\text{dof}}$ erreicht wurde, oder aber nach 100 Iteration. Dies kann sehr praktisch sein, wenn man viele verschiedene Fits durchführen muss und die Startwerte nicht sehr gut kennt (wie z.B. im F-Praktikumsversuch Halbleiterdetektoren). Erfahrungsgemäß findet `wnonlinfit` die wahren Werte auch oft bei recht schlechten Startwerten, bei denen der erste Versuch nicht funktioniert. Man kann die zufällige Variation der Startwerte auch deaktivieren, indem man die Toleranz für $\frac{\chi^2}{\text{dof}}$ einfach sehr hoch setzt. Siehe dazu Kapitel 7

4.2.1 Beispiel

In diesem Beispiel soll mit gegebenen Daten des Spektrums eines Schwarzkörperstrahlers dessen Temperatur bestimmt werden und überprüft werden, ob es sich tatsächlich um einen Schwarzkörperstrahler handelt. Das Plancksche Strahlungsgesetz lautet nun

$$S(E_{\mathrm{ph}}) = A_0 \cdot \frac{E_{\mathrm{ph}}^3}{\exp\left(E_{\mathrm{ph}}\beta\right)-1} \quad \text{mit} \quad \beta = \frac{1}{k_b T}$$

Mit `wnonlinfit` kann diese Funktion nun direkt gefittet werden. Möglicherweise sollte man sich die Funktion und Daten plotten lassen um geeignete Startwerte zu finden, sofern man keine physikalischen Schätzungen besitzt. In diesem Beispiel ist die Energie in eV in den Daten gegeben, die y-Werte sind in Counts also einheitenlos. Im weiteren soll nun der Code behandelt werden.

Listing 4.1: Beispiel eines nichtlinearen Fits mit wnonlinfit

```
1  clear;
2
3  % First the data has to be loaded
4  % Expansion in three variables is optional
5  load('nonlinfit.dat')
6  x = nonlinfit(:,1);
7  y = nonlinfit(:,2);
8  yerr = nonlinfit(:,3);
9
10 % you need to specify the function you want to use for fitting
11 % the syntax is
12 % fitfunc(vector of xvalues,vector of fitparameters) = @(x,c) ...
13 fitfunc=@(x,cv) cv(1)*x.^3./(exp(x.*cv(2))-1);
14
15 % you also need a first guess for the fitparameters:
16 c0=[2000 2];
17
18
19 % normally the input data has to be vector of xvalues, vector of yvalues,
20 % vector of errors. If you don't have errors you can just put a 1 (or
21 % arbitrary value) instead of a vector
22
23
24 %%%%% Optional input %%%%%
25 textposition=[4 600];     % define position of legend
26
27 % define header of legend
28 headercell={'Nonlinear Fit'  '$f(E_{\mathrm{ph}})=A_0 \cdot \frac{E_{\mathrm{ph
       }}^3}{\exp{(E_{\mathrm{ph}} \beta)}{-}1}$' ''};
29 % define axis labels and fitparameter names
30 mylabel={'Photon energy $E_{\mathrm{ph}}$ [eV]','Photon Counts','$A_0$','$\beta$'};
31 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
32
33
34 % actual execution of wnonlinfit. See "help wnonlinfit" for more options.
35 [c cerr Kov Chi]=wnonlinfit(x,y,yerr,fitfunc,c0...          % main arguments
36      ,'label',mylabel,'position',textposition,'header',headercell,'errprec',2,'
           contourplot',{[1 2]});  % optional arguments
37
38 Tres = 1/c(2)/1.3807e-23*1.6022e-19;
39 Treserr = 1/c(2)^2*cerr(2)/1.3807e-23*1.6022e-19;
40 titlestr = sprintf('Blackbody radiation spectrum @$T=%4.0f \\pm %2.0f$ K',[Tres
       Treserr]);
41 title(titlestr,'interpreter','latex','fontsize',16)
42
43 print -dpdf -cmyk result_nonlinfit.pdf
```

```
44 figure(3)
45 print -dpdf -cmyk contour_nonlinfit.pdf
```

Zeile 1: Zunächst sollte man alle im Speicher befindlichen Variablen mit **clear** löschen um möglichen Fehlern vorzubeugen.

Zeile 5: Laden der in der Datei nonlinfit.dat gespeicherten Daten.

Zeile 6-8: Umspeichern der Daten von einer $N \times 3$ in drei $N \times 1$ Matrizen (auch manchmal Vektoren genannt).

Zeile 13: Definition der Funktion, die zum Fitten verwendet werden soll als fitfunc. @(x,betav) definiert dabei, dass die Funktion von x und betav abhängen soll. Die Formel definiert nun die ausgegebenen Werte. Die Funktion soll nun wenn x die Dimension $N \times 1$ besitzt auch eine Variable der Dimension $N \times 1$ ausgeben. betav definiert einen Vektor der Fitparameter, die einfach elementweise verwendet werden können.

Zeile 16: Startwerte für die Fitparameter.

Zeile 25: Variable für die Position der Legende im Plot. Der erste Eintrag definiert die horizontale Position der zweite die vertikale Position.

Zeile 28-30: Definition zweier Variabler für den Text im Plot. Headercell definiert später die Überschrift in der Legende. Dabei steht jeder neue Eintrag für eine Zeile. mylabel wird für die Achsenbeschriftung und Benennung der Fitparameter in der Legende verwendet.

Zeile 35: Aufruf von wnonlinfit. Dabei werden die vorher definierten Variablen verwendet und die Ergebnisse in [c cerr Kov Chi] gespeichert. Desweiteren werden zwei Plots erstellt. Für die verwendeten Optionen siehe Kapitel 7.

Zeile 38-39: Umrechnung des Ergebnisses in eine Temperatur in Kelvin.

Zeile 40-41: Fügt dem Plot eine Überschrift hinzu in der das Ergebnis, also die Temperatur des Strahlers in Kelvin, ausgegeben wird.

Zeile 43: Speichert den Plot in eine pdf-Datei.

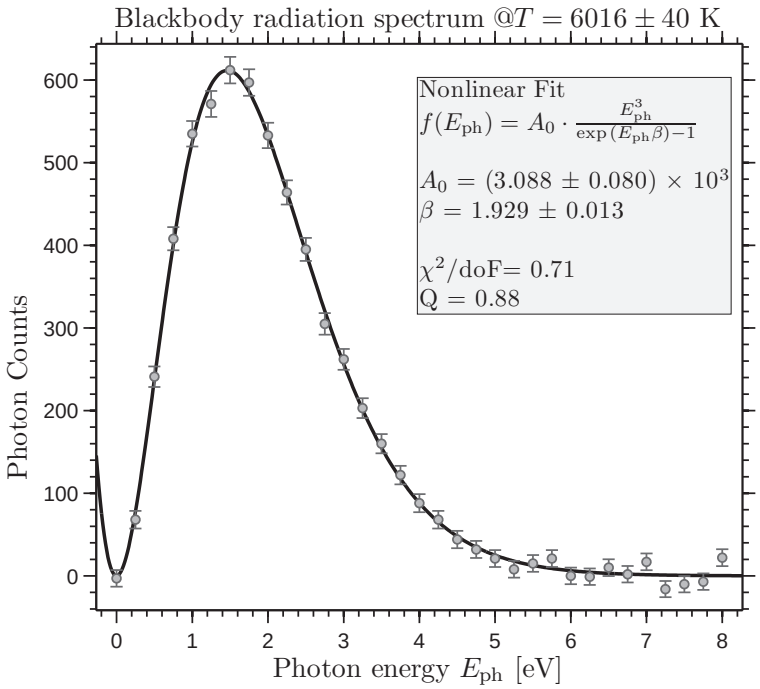

Abbildung 4.2.1: Fit von generierten Daten eines Schwarzkörperstrahlungsspektrums mit wnonlinfit

5 Theorie: Lineare Regression

Im Allgemeinen lässt sich χ^2_{min} nicht analytisch bestimmen, sondern muss numerisch berechnet werden. Falls aber

$$f(x|\boldsymbol{\theta}) = \sum_{j=1}^{m} \theta_j h_j(x) \tag{5.0.1}$$

mit linear unabhängigen Funktionen h_j, dann reduziert sich die Regression samt Bestimmung der Konfidenzintervalle auf ein paar wenige Matrixgleichungen. Die folgende Rechnung orientiert sich sowohl am CP-Skript [1] und dem *Particle Data Booklet* [2].

Unser erstes Ziel ist es, die optimalen Parameter $\hat{\theta}$ zu finden, d.h. die Parameter, für die χ^2 in Gl. (3.2.3) minimal wird. Dazu ist es notwendig, dass die partiellen Ableitungen nach allen θ_k von χ^2 in $\hat{\boldsymbol{\theta}}$ verschwinden:

$$\frac{\partial}{\partial \theta_k} \chi^2 \bigg|_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}} = 0 \qquad \forall k, \tag{5.0.2}$$

also

$$\begin{aligned} \frac{\partial}{\partial \theta_k} \left[\sum_{i=1}^{N} \frac{(y_i - f(x_i|\boldsymbol{\theta}))^2}{\sigma_i^2} \right]_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}} &= -\sum_{i=1}^{N} \frac{2h_k(x_i)}{\sigma_i^2} \, (y_i - f(x_i|\hat{\boldsymbol{\theta}})) = 0 \\ \Leftrightarrow \quad \sum_{i=1}^{N} \frac{h_k(x_i) y_i}{\sigma_i^2} &= \sum_{i=1}^{N} \frac{h_k(x_i)}{\sigma_i^2} \, f(x_i|\hat{\boldsymbol{\theta}}). \end{aligned} \tag{5.0.3}$$

Wenn wir jetzt die folgenden Matrizen definieren:

$$H = \begin{pmatrix} h_1(x_1) & h_2(x_1) & \dots \\ h_1(x_2) & h_2(x_2) & \\ \vdots & & \ddots \end{pmatrix} \qquad V = \begin{pmatrix} \sigma_1^2 & 0 & \dots & 0 \\ 0 & \sigma_2^2 & & \\ \vdots & & \sigma_3^2 & \\ 0 & & & \ddots \end{pmatrix}. \tag{5.0.4}$$

Dann ist

$$\begin{pmatrix} f(x_1|\boldsymbol{\theta}) \\ f(x_2|\boldsymbol{\theta}) \\ \vdots \end{pmatrix} = H\boldsymbol{\theta} \qquad \text{und} \qquad \sum_{i=1}^{N} \frac{h_k(x_i)}{\sigma_i^2} \, f(x_i|\hat{\boldsymbol{\theta}}) = \left(H^T V^{-1} H \hat{\boldsymbol{\theta}} \right)_k, \tag{5.0.5}$$

wobei H^T die transponierte Matrix zu H ist. Man kann Gl. (5.0.3) deshalb so zusammenfassen:

$$\underbrace{H^T V^{-1} \mathbf{y}}_{=:\mathbf{g}} = \underbrace{H^T V^{-1} H}_{=:U^{-1}} \hat{\boldsymbol{\theta}} \qquad \Leftrightarrow \qquad U\mathbf{g} = \hat{\boldsymbol{\theta}}, \tag{5.0.6}$$

bzw.

$$\hat{\boldsymbol{\theta}} = \underbrace{(H^T V^{-1} H)^{-1} H^T V^{-1}}_{=:D} \mathbf{y} \tag{5.0.7}$$

Nachdem wir nun $\hat{\boldsymbol{\theta}}$ bestimmt haben, wollen wir die Varianzen σ_{θ_k} bestimmen. Das können wir z.B. mit Gauß'scher Fehlerfortpflanzung machen:

$$\sigma_{\theta_k}^2 = \sum_{i=1}^{N} \sigma_i^2 \left(\frac{\partial \theta_k}{\partial y_i} \right)^2 .$$

Die partiellen Ableitungen bekommen wir über den (altbekannten) Trick mit dem Differential:

$$\delta\theta_k \;=\; \delta(D\mathbf{y})_k \;=\; \sum_{j=1}^{N} d_{kj} \delta y_i \;\stackrel{!}{=}\; \sum_{j=1}^{N} \left(\frac{\partial \theta_k}{\partial y_i} \right) \delta y_i .$$

Also ist

$$\sigma_{\theta_k}^2 = \sum_{i=1}^{N} \sigma_i^2 d_{ki}^2 = (DVD^T)_{kk}.$$

Wenn wir nun noch auswerten, was DVD^T ist, so stellen wir fest:

$$\begin{aligned} DVD^T &= [(H^T V^{-1} H)^{-1} H^T V^{-1}] V [(H^T V^{-1} H)^{-1} H^T V^{-1}]^T \\ &= (H^T V^{-1} H)^{-1} H^T V^{-1} V V^{-1} H (H^T V^{-1} H)^{-1} \\ &= (H^T V^{-1} H)^{-1} = U. \end{aligned}$$

Wir wissen also, dass auf der Diagonalen von U die Quadrate der Unsicherheiten σ_{θ_k} stehen. Welche Informationen enthält U noch?

5.1 Kovarianzmatrix

Die Matrix U ist die sogenannte Kovarianzmatrix:

$$U = \begin{pmatrix} \sigma_{\theta_1}^2 & \sigma_{\theta_1\theta_2}^2 & \cdots \\ \sigma_{\theta_1\theta_2}^2 & \sigma_{\theta_2}^2 & \\ \vdots & & \ddots \end{pmatrix} . \tag{5.1.1}$$

Was das bedeutet, schauen wir uns nun an. Zunächst schreiben wir Gl. (3.2.3) um:

$$\chi^2 = (\mathbf{y} - H\boldsymbol{\theta})^T V^{-1} (\mathbf{y} - H\boldsymbol{\theta}). \tag{5.1.2}$$

Weil V^{-1} symmetrisch ist, können wir dies wie eine binomische Formel umformen:

$$\chi^2(\boldsymbol{\theta}) = \mathbf{y}^T V^{-1}\mathbf{y} \;-\; 2\mathbf{y}^T V^{-1} H\boldsymbol{\theta} \;+\; (H\boldsymbol{\theta})^T V^{-1} H\boldsymbol{\theta} \tag{5.1.3}$$

Für den Spezialfall $\boldsymbol{\theta} = \hat{\boldsymbol{\theta}}$ gilt:

$$\begin{aligned}
\chi^2_{\min} = \chi^2(\hat{\boldsymbol{\theta}}) &= \mathbf{y}^T V^{-1}\mathbf{y} \;-\; 2\mathbf{y}^T V^{-1} H\hat{\boldsymbol{\theta}} \;+\; (H\hat{\boldsymbol{\theta}})^T V^{-1} H\hat{\boldsymbol{\theta}} \\
&= \mathbf{y}^T V^{-1}\mathbf{y} \;-\; 2\mathbf{y}^T D^T H^T V^{-1} H\hat{\boldsymbol{\theta}} \;+\; (H\hat{\boldsymbol{\theta}})^T V^{-1} H\hat{\boldsymbol{\theta}} \\
&= \mathbf{y}^T V^{-1}\mathbf{y} \;-\; 2(H\hat{\boldsymbol{\theta}})^T V^{-1} H\hat{\boldsymbol{\theta}} \;+\; (H\hat{\boldsymbol{\theta}})^T V^{-1} H\hat{\boldsymbol{\theta}} \\
&= \mathbf{y}^T V^{-1}\mathbf{y} \;-\; (H\hat{\boldsymbol{\theta}})^T V^{-1} H\hat{\boldsymbol{\theta}}
\end{aligned}$$

Für allgemeines $\boldsymbol{\theta}$ machen wir in Gl. (5.1.3) eine passende Ergänzung:

$$\begin{aligned}
\chi^2(\boldsymbol{\theta}) &= (H\boldsymbol{\theta})^T V^{-1} H\boldsymbol{\theta} \;-\; 2\mathbf{y}^T V^{-1} H\boldsymbol{\theta} \;+\; (H\hat{\boldsymbol{\theta}})^T V^{-1} H\hat{\boldsymbol{\theta}} \;+\; \chi^2_{\min} \\
&= (H\boldsymbol{\theta} - H\hat{\boldsymbol{\theta}})^T V^{-1} (H\boldsymbol{\theta} - H\hat{\boldsymbol{\theta}}) \;+\; \chi^2_{\min} \\
&= (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^T U^{-1} (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}) \;+\; \chi^2_{\min}
\end{aligned} \tag{5.1.4}$$

Was bedeutet diese Gl. (5.1.4) anschaulich? Zur Vereinfachung nehmen wir kurz an, dass U diagonal ist - was es meistens *nicht* ist. Dann bedeutete Gl. (5.1.4):

$$\chi^2(\boldsymbol{\theta}) = \sum_{k=1}^{m} \frac{(\theta_k - \hat{\theta}_k)^2}{\sigma^2_{\theta_k}} + \chi^2_{\min}.$$

Setzt man für ein θ_k stattdessen $\hat{\theta}_k \pm \sigma_{\theta_k}$ und für die anderen θ den optimalen Wert ein, so erhalt man:

$$\chi^2 \left[\begin{pmatrix} \hat{\theta}_1 \\ \hat{\theta}_2 \\ \vdots \\ \hat{\theta}_{k-1} \\ \hat{\theta}_k \pm \sigma_{\theta_k} \\ \hat{\theta}_{k+1} \\ \vdots \end{pmatrix} \right] = \frac{\sigma^2_{\theta_k}}{\sigma^2_{\theta_k}} + \chi^2_{\min} = 1 + \chi^2_{\min}.$$

Das gilt für beliebige k. Würde man sich die Menge aller Vektoren $\boldsymbol{\theta}$ anschauen, wo $\chi^2(\boldsymbol{\theta}) = \chi^2_{\min} + 1$ ist, so würde diese einen Ellipsoiden darstellen, im zweidimensionalen Fall also eine Ellipse wie in Abb. 3.2.2. Solange U diagonal ist, wären ihre Hauptachsen parallel zu den Koordinatenachsen. Im anderen, sehr viel häufiger auftretenden Fall liegt die Ellipse (bzw. der Ellipsoid) schief im Koordinatensystem und es entstehen Abhängigkeiten zwischen den Fitparametern - die Fitparameter sind **korreliert**. Deshalb hängen die Nichtdiagonalelemente σ^2_{ij} von U auch mit den **Korrelationskoeffizienten** zusammen und heißen **Kovarianzen**.

Kovarianzen. Um die praktische Bedeutung von Kovarianzen erklären zu können, müssen wir erst einmal auf die stochastische eingehen. Als **Varianz** σ^2_X einer Zufallsvariable X hatten wir definiert:

$$\sigma^2_X = \mathbb{E}(X^2) - (\mathbb{E}X)^2.$$

Die Kovarianz betrifft nun zwei Zufallsvariablen, z.B. X und Y:

$$\sigma^2_{XY} = \mathbb{E}(XY) - (\mathbb{E}X)(\mathbb{E}Y) = \mathbb{E}\left((X - \mathbb{E}X)(Y - \mathbb{E}Y)\right). \tag{5.1.5}$$

Man sagt, X und Y sind **unkorreliert**, wenn $\mathbb{E}(XY) = (\mathbb{E}X)(\mathbb{E}Y)$. Dann ist $\sigma^2_{XY} = 0$.

Achtung! Unkorreliertheit ist nicht äquivalent zu Unabhängigkeit[a]! Zwar impliziert Unabhängigkeit Unkorreliertheit, nicht aber andersherum.

[a] Zwei Zufallsvariablen X und Y heißen unahängig, falls für jede zwei *vernünftig gewählte* Mengen A und B aus dem Ereignisraum gilt: $P\left[(X \in A) \cap (Y \in B)\right] = P(X \in A)P(Y \in B)$.

Was bedeutet nun also $\sigma^2_{\theta_i\theta_j}$? Wir hatten als zusammengesetzte Wahrscheinlichkeitsdichte $\varphi(\mathbf{y}|\boldsymbol{\theta})$ der Messwerte $\mathbf{y}$ unter der Voraussetzung der Werte der Fitparameter $\boldsymbol{\theta}$ gefunden (siehe Gl. 3.2.1):

$$\varphi(\mathbf{y}|\boldsymbol{\theta}) = \prod_{i=1}^{N} \mathcal{N}_{f(x_i|\boldsymbol{\theta}),\sigma_i}(y_i) \propto e^{-\chi^2/2},$$

wobei das Proportionalitätszeichen hier bedeutet, dass der Normierungsfaktor weglassen wurde. Man kann die Abhängigkeit von $\mathbf{y}$ und $\boldsymbol{\theta}$ umkehren, sodass $\boldsymbol{\theta}$ die eigentlich freie Variable ist und $\mathbf{y}$ der Parameter. Das ist ja eigentlich auch die Situation, die vorliegt, denn unsere Messwerte sind fest und wir variieren $\boldsymbol{\theta}$. Dann erhalten wir eine Verteilung für $\boldsymbol{\theta}$:

$$\begin{aligned}\psi(\boldsymbol{\theta}|\mathbf{y}) \propto e^{-\chi^2/2} &= \exp\left[-\frac{1}{2}\left((\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^T U^{-1} (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}) + \chi^2_{\min}\right)\right] \\ &\propto \exp\left[-\frac{1}{2}(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^T U^{-1} (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})\right].\end{aligned} \tag{5.1.6}$$

Die $\mathbf{y}$-Abhängigkeit steckt im $\hat{\boldsymbol{\theta}}$. Man kann ausrechnen, dass die Verteilung mit Normierungsfaktor so aussieht:

$$\psi(\boldsymbol{\theta}|\mathbf{y}) = \frac{1}{(2\pi)^{m/2}\sqrt{\det U}} \exp\left[-\frac{1}{2}(\boldsymbol{\theta}-\hat{\boldsymbol{\theta}})^T U^{-1}(\boldsymbol{\theta}-\hat{\boldsymbol{\theta}})\right] \tag{5.1.7}$$

Wir wollen nun untersuchen, was

$$\mathbb{E}(\theta_j\theta_k) - (\mathbb{E}\theta_j)(\mathbb{E}\theta_k) = \mathbb{E}\left((\theta_j - \mathbb{E}\theta_j)(\theta_k - \mathbb{E}\theta_k)\right)$$

ist. Wir wissen schon, dass:

$$\mathbb{E}\theta_j = \hat{\theta}_j.$$

Sei $\boldsymbol{\vartheta} = \boldsymbol{\theta} - \hat{\boldsymbol{\theta}}$. Man *kann* zeigen:

$$\begin{aligned}
\mathrm{K}(\theta_j,\theta_k) &= \mathbb{E}\left[(\theta_j - \mathbb{E}\theta_j)(\theta_k - \mathbb{E}\theta_k)\right] \\
&= \int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty}(\theta_j-\hat{\theta}_j)(\theta_k-\hat{\theta}_k)\ \psi(\boldsymbol{\theta}|\mathbf{y})\ d\theta_1\ldots d\theta_m \\
&= \frac{1}{(2\pi)^{m/2}\sqrt{\det U}}\int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty}\vartheta_j\vartheta_k\ \exp\left[-\frac{1}{2}\boldsymbol{\vartheta}^T U^{-1}\boldsymbol{\vartheta}\right]\ d\vartheta_1\ldots d\vartheta_m.
\end{aligned}$$

Sei weiterhin die Matrix O so, dass $W = OU^{-1}O^T$ diagonal ist. Es ist $O^T = O^{-1}$, weil U symmetrisch ist. Sei weiterhin $\boldsymbol{\eta} = O\boldsymbol{\vartheta}$. Dann ist

$$\boldsymbol{\vartheta}^T U^{-1}\boldsymbol{\vartheta} \ = \ \boldsymbol{\vartheta}^T O^T O U^{-1} O^T O\boldsymbol{\vartheta} \ = \ (O\boldsymbol{\vartheta})^T W(O\boldsymbol{\vartheta}) \ = \ \boldsymbol{\eta}^T W\boldsymbol{\eta}$$

und die Funktionaldeterminante bei der Transformation $\boldsymbol{\vartheta} \to \boldsymbol{\eta}$ gleich $\det O = 1$ und deshalb:

$$\mathrm{K}(\theta_j,\theta_k) = \frac{1}{(2\pi)^{m/2}\sqrt{\det U}}\int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty}(O^T\boldsymbol{\eta})_j(O^T\boldsymbol{\eta})_k\ \exp\left[-\frac{1}{2}\boldsymbol{\eta}^T W\boldsymbol{\eta}\right]\ d\eta_1\ldots d\eta_m.$$

Sei $W = \mathrm{diag}(\lambda_1^{-1},\ldots,\lambda_m^{-1})$, dann $\det U = (\det W)^{-1} = \lambda_1\cdot\ldots\cdot\lambda_m$ und

$$\begin{aligned}
\mathrm{K}(\theta_j,\theta_k) = \frac{1}{(2\pi)^{m/2}\sqrt{\lambda_1\cdot\ldots\cdot\lambda_m}}\int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty}\left(\sum_l O^T_{jl}\eta_l\right)\left(\sum_q O^T_{kq}\eta_q\right)\times \\
\times\prod_{k=1}^{m}\exp\left[-\frac{1}{2}\lambda_k^{-1}\eta_k^2\right]\ d\eta_1\ldots d\eta_m \\
= \int_{-\infty}^{\infty}\cdots\int_{-\infty}^{\infty}\frac{\left(\sum_l O^T_{jl}O^T_{kl}\eta_l^2\right)\ \prod_{k=1}^{m}\exp\left[-\frac{1}{2}\lambda_k^{-1}\eta_k^2\right]}{(2\pi)^{m/2}\sqrt{\lambda_1\cdot\ldots\cdot\lambda_m}}\ d\eta_1\ldots d\eta_m
\end{aligned}$$

$$= \left(\sum_l O^T_{jl} O^T_{kl} \lambda_l \right) = \left(\sum_l O^T_{kl} \lambda_l O_{lj} \right) = \left(O^T W^{-1} O \right)_{kj} = (U)_{kj} = \sigma^2_{\theta_j \theta_k}.$$

Zwischendurch haben wir ausgenutzt, dass alle Terme, wo η_l nicht quadratisch auftaucht, verschwinden.

Die Kovarianz $\sigma^2_{\theta_j \theta_k}$ aus der Kovarianzmatrix U ist also genau auch die stochastische Kovarianz zwischen θ_j und θ_k:

$$\sigma^2_{\theta_j \theta_k} = \mathbb{E}(\theta_j \theta_k) - (\mathbb{E}\theta_j)(\mathbb{E}\theta_k). \tag{5.1.8}$$

5.2 Fehlerfortpflanzung

Nehmen wir nun an, wir wollten mit den Fitparametern $\hat{\boldsymbol{\theta}}$ weiterrechnen und eine Größe Z bestimmen, die von θ_j und θ_k abhängt. Dann gilt für die Varianz σ^2_Z mittels einer Taylor-Entwichlung:

$$\begin{aligned}
\sigma^2_Z &= \mathbb{E}(Z^2) - (\mathbb{E}Z)^2 = \mathbb{E}\left[(Z - \mathbb{E}Z)^2 \right] \\
&\approx \mathbb{E}\left[\left(Z(\hat{\theta}_j, \hat{\theta}_k) + \left(\frac{\partial Z}{\partial \theta_j} \right) (\theta_j - \hat{\theta}_j) + \left(\frac{\partial Z}{\partial \theta_k} \right) (\theta_k - \hat{\theta}_k) - \mathbb{E}Z \right)^2 \right]
\end{aligned}$$

Man nimmt nun zusätzlich an, dass

$$Z(\hat{\theta}_j, \hat{\theta}_k) = Z\left(\mathbb{E}\theta_j, \mathbb{E}\theta_k \right) \approx \mathbb{E}\left(Z(\theta_j, \theta_k) \right).$$

Diese Annahme ist sehr stark! Im Allgemeinen darf man *nicht* annehmen, dass $Z\left(\mathbb{E}\theta_j, \mathbb{E}\theta_k \right) \approx \mathbb{E}\left(Z(\theta_j, \theta_k) \right)$. Gehen wir nur einmal davon aus, dass $Z = \theta_j \theta_k$. Dann würde die obige Annahme bedeutet, dass die Kovarianz zwischen θ_j und θ_k gleich Null ist, wovon wir hier explizit nicht ausgehen. Dennoch ist die obige Annahme nötig, um auf das Gauß'sche Fehlerfortpflanzungsgesetz zu kommen:

$$\begin{aligned}
\sigma^2_Z &\approx \mathbb{E}\left[\left(\left(\frac{\partial Z}{\partial \theta_j} \right) (\theta_j - \hat{\theta}_j) + \left(\frac{\partial Z}{\partial \theta_k} \right) (\theta_k - \hat{\theta}_k) \right)^2 \right] \\
&= \mathbb{E}\left[\left(\frac{\partial Z}{\partial \theta_j} \right)^2 (\theta_j - \hat{\theta}_j)^2 + \left(\frac{\partial Z}{\partial \theta_k} \right)^2 (\theta_k - \hat{\theta}_k)^2 + 2 \left(\frac{\partial Z}{\partial \theta_j} \right) \left(\frac{\partial Z}{\partial \theta_k} \right) (\theta_j - \hat{\theta}_j)(\theta_k - \hat{\theta}_k) \right] \\
&= \left(\frac{\partial Z}{\partial \theta_j} \right)^2 \mathbb{E}\left[(\theta_j - \hat{\theta}_j)^2 \right] + \left(\frac{\partial Z}{\partial \theta_j} \right)^2 \mathbb{E}\left[(\theta_k - \hat{\theta}_k)^2 \right] \\
&\quad + 2 \left(\frac{\partial Z}{\partial \theta_j} \right) \left(\frac{\partial Z}{\partial \theta_k} \right) \mathbb{E}\left[(\theta_j - \hat{\theta}_j)(\theta_k - \hat{\theta}_k) \right] \\
&= \left(\frac{\partial Z}{\partial \theta_j} \right)^2 \sigma^2_{\theta_j} + \left(\frac{\partial Z}{\partial \theta_k} \right)^2 \sigma^2_{\theta_k} + 2 \left(\frac{\partial Z}{\partial \theta_j} \right) \left(\frac{\partial Z}{\partial \theta_k} \right) \sigma^2_{\theta_j \theta_k}.
\end{aligned}$$

Man benötigt die Kovarianz also bei der Fehlerfortpflanzung, wenn zwei Größen korreliert sind, wie hier θ_j und θ_k.

5.3 Konfidenzintervalle bei nichtlinearer Regression

Nachdem man im nichtlinearen Fall die optimalen $\hat{\boldsymbol{\theta}}$ durch eine Minimierungsroutine gefunden hat, braucht man zusätzlich noch die Unsicherheiten $\sigma_{\boldsymbol{\theta}}$. Um diese zu finden, bietet es sich an, das Problem zu linearisieren. Dazu approximiert man die Fitfunktion $f(x|\boldsymbol{\theta})$ durch

$$f(x_i|\boldsymbol{\theta}) \approx f(x_i|\hat{\boldsymbol{\theta}}) + \sum_j F_{ij}(\theta_j - \hat{\theta_j}) \qquad \text{mit} \quad F_{ij} = \frac{\partial f(x_i|\boldsymbol{\theta})}{\partial \theta_j}.$$

Dies setzt man nun in Gl. (3.2.3) ein und erhält

$$\chi^2(\theta) \approx \sum_i \frac{\left(y_i - f(x_i|\hat{\boldsymbol{\theta}}) + \sum_j F_{ij}(\theta_j - \hat{\theta_j})\right)^2}{\sigma_i^2}.$$

Dies entspricht dem in Kapitel 5 beschriebenen linearen Fall, mit der Ersetzung $H \to F$ und $y_i \to (y_i - f(x_i|\hat{\boldsymbol{\theta}}))$. Führt man die Rechnung analog dazu aus, kommt man auf

$$\boldsymbol{\theta} - \hat{\boldsymbol{\theta}} = \tilde{D}(\mathbf{y} - \mathbf{f}(\hat{\boldsymbol{\theta}})) \tag{5.3.1}$$

$$\text{mit: } (\mathbf{f}(\hat{\boldsymbol{\theta}}))_i = f(x_i|\hat{\boldsymbol{\theta}}) \quad \text{und} \quad \tilde{D} = (F^T V^{-1} F)^{-1}(F^T V^{-1})$$

Für die Unsicherheiten benutzt man analog zu Kapitel 5:

$$\delta\boldsymbol{\theta} - \delta\hat{\boldsymbol{\theta}} = \tilde{D}(\delta\mathbf{y} - \delta\mathbf{f}(\hat{\boldsymbol{\theta}})) = \tilde{D}(\delta\mathbf{y} - F\hat{\boldsymbol{\theta}})$$

Es ist

$$\tilde{D}F = (F^T V^{-1} F)^{-1}(F^T V^{-1} F) = \mathbb{I}$$

und damit:

$$\delta\boldsymbol{\theta} = \tilde{D}\delta\mathbf{y}.$$

Daraus folgt wie im linearen Fall für die Kovarianzmatrix $\tilde{U}$:

$$\tilde{U} = (F^T V^{-1} F)^{-1}. \tag{5.3.2}$$

Auf der Diagonalen dieser Matrix findet man wiederum die Varianzen $\sigma_{\theta_i}^2$ und die Nichtdiagonalelemente geben den entsprechenden Kovarianzen an.

6 Numerik: Lineare Regression

6.1 Linearer Fit mit wlinfit

Die Syntax für einen Aufruf von wlinfit lautet

$$\left[\boldsymbol{\theta},\ \boldsymbol{\theta}_{\text{err}},\ f(x),\ U_{\text{kov}},\ \chi^2/\text{doF}\right] = \text{wlinfit}\left(\mathbf{x}, \mathbf{y}, \mathbf{y}_{\text{err}}, \{@(x)f_1(x),\ @(x)f_2(x), ...\}\right) \tag{6.1.1}$$

Die Eingabeparameter sind dabei:

- $\mathbf{x}$: Vektor der X-Werte
- $\mathbf{y}$: Vektor der Y-Werte
- $\delta\mathbf{y}$: Vektor der Unsicherheiten der Y-Werte
- $\{@(x)f_1(x),\ @(x)f_2(x), ...\}$: sog. Cell mit den zu fittenden Funktionen (Bsp: $\{@(x)\cos(x),\ @(x)\sin(x)\}$)

Die Ausgabewerte sind:

- $\boldsymbol{\theta}$: Vektor der Koeffizienten des Fits, also die Ergebnisse.
- $\boldsymbol{\theta}_{\text{err}}$: Vektor der Unsicherheiten $\sigma_{\mathbf{x}}$ von $\boldsymbol{\theta}$
- $f(x)$: Die resultierende Funktion, d.h. die entsprechenden c_i mit den zu fittenden Funktionen multipliziert. Diese kann in Matlab direkt weiterverwendet werden durch Aufruf von z.B. $\mathbf{u}_{\text{theo}} = f(\mathbf{x})$
- U_{kov}: Kovarianzmatrix.
- χ^2/doF: Der χ^2-Wert geteilt durch die Anzahl der Freiheitsgrade. Dieser sollte nahe bei 1 liegen.

Wenn man mit einem Polynom n-ter Ordnung fitten will kann man auch die Option 'polyfit' verwenden und muss die Funktionen nicht direkt übergeben. Dazu ruft man `wlinfit` auf, ohne Funktionen zu übergeben. Allerdings darf man allerdings das Feld nicht einfach leer lassen, sondern muss {}, also eine leere Cell, einsetzen. Dann setzt man die Option 'polyfit' auf den gewünschten Grad des Polynoms. Dies sieht für ein Polynom erster Ordnung dann wie folgt aus

$$\left[\boldsymbol{\theta},\ \boldsymbol{\theta}_{\text{err}},\ f(x),\ U_{\text{kov}},\ \chi^2/\text{DoF}\right] = \text{wlinfit}\left(\mathbf{x}, \mathbf{y}, \delta\mathbf{y}, \{\}, \text{'polyfit'}, 1\right) \tag{6.1.2}$$

Daraufhin werden zwei Plots erstellt. Der erste ist in Abb. 6.1.1 zu sehen. Darin sieht man die gemessenen Werte in grau mit Fehlern, die resultierende Funktion in schwarz und eine Tabelle mit den Ergebnissen. Der zweite Plot stellt die Residuen dar, welche durch die Formel

$$\mathrm{res}_i = \frac{x_i - f(x_i; \tau, U_0)}{\sigma_{U,i}}$$

gegeben sind. Um Plots als pdf zu speichern geht man wie folgt vor

```
print -dpdf filenameplot.pdf
figure(2)
print -dpdf filenameresiduen.pdf
```

6.1.1 Beispiel

In diesem Beispiel soll das Entladen einen Kondensators behandelt werden. Dabei wurde die Spannung U als Funktion der Zeit t aufgenommen und als $N \times 3$ Matrix abgespeichert (Listing 6.2). Dabei steht die Zeit in der ersten, die Spannung in der zweiten und die Unsicherheit der Spannung in der dritten Spalte. Der bekannte funktionale Zusammenhang ist dabei

$$U(t) = U_0 \cdot e^{-\frac{t}{\tau}}$$

mit den zu bestimmenden Parametern U_0 und τ. Dies ist noch nicht linear kann aber durch Logarithmieren in einen linearen Fit umgewandelt werden.

$$\log(U) = \log(U_0) - \frac{1}{\tau} \cdot t = c_1 + c_2 \cdot t \quad \text{mit} \quad c_1 = \log(U_0) \quad \text{und} \quad c_2 = -\frac{1}{\tau}$$

Es muss außerdem die Unsicherheit $\delta \log(U)$ berechnet werden. Diese ergibt sich mit der Gaußschen Fehlerfortpflanzung zu

$$\delta \log(U) = \frac{\delta U}{U}$$

Nach dieser Vorbereitung kann mit dem Code begonnen werden. Ein lauffähiger Code ist in Listing 6.1 zu finden und soll hier erläutert werden.

Listing 6.1: Beispiel eines linearen Fits mit wlinfit

```
clear                % Loeschen aller im Speicher befindlichen Variablen

load linfit.dat      % Laden der Messwerte als N x 3 Matrix

t = linfit(:,1);     % Speichern der jeweiligen Spalten der Matrix
U= linfit(:,2);      % als Vektoren
Uerr = linfit(:,3); %

[c cerr resultfunc Kov] = wlinfit(t,log(U),Uerr./U,{},'polyfit',1,'label',{'Zeit t
    [s]','Ln(Spannung [V])', '$\mathrm{ln}(U_0)$ ', '$\beta$'},'header','$\mathrm{
    ln}(U(t))=\mathrm{ln}U_0+\beta t $','contourplot',{[1 2]},'print',true,'
    position',[1.4e-3 0]);

print -dpdf polyfit.pdf

```

```
13 tau = -1/c(2);
14 dtau = cerr(2)/c(2)^2;
15
16 U0 = exp(c(1));
17 dU0 = exp(c(1))*cerr(1);
18 fprintf(' tau=%1.3e +- %1.3e\n U_0=%1.3e +- %1.3e\n',[tau dtau U0 dU0]);
```

Zeile 1: Zunächst sollte man alle im Speicher befindlichen Variablen mit **clear** löschen um möglichen Fehlern vorzubeugen.

Zeile 3: In Zeile 3 werden mit dem Befehl **load** die Daten, die in einem ASCII file (Listing 6.2) hinterlegt sind, unter dem Namen `linfit` in den Speicher geladen.

Zeile 5-7: In Zeile 5 bis 7 werden die 3 Spalten der Matrix *linfit* als eigene Variablen abgespeichert.

Zeile 9: Nun kann in Zeile 9 die eigentlich Routine **wlinfit** aufgerufen werden. Dabei werden gemäß Gl. 6.1.2 die x-Werte t, die y-Werte $\mathbf{log}(U)$ mit den Unsicherheiten U_{err} mit einem Polynom erster Ordnung gefittet, was mit der Option 'polyfit' spezifiziert wurde. Weiterhin wurde mit 'label' die Beschriftung der Achsen und die Benennung der Variablen im Fit definiert. Dazu übergibt man eine Cell mit den entsprechenden Bezeichnungen in der Form {'x-achse' 'y-achse' '1. Variable' '2.Variable'}. Wenn man wie hier keine Variablennamen spezifiziert, werden sie einfach c_1, c_2 usw. genannt. Die letzte Option 'header' definiert die Überschrift in der Legende. Mehr zu Optionen findet man in Kapitel 7
Wichtig: Alle Strings müssen in Latex eingegeben werden. Dabei befindet man sich im Textmodus. Will man in den mathematische Modus, geht das wie üblich mit $$

Zeile 11: Mit **print** kann man Bilder als Dateien speichern. Die Option -dpdf spezifiziert das format (siehe **help print** für mehr Optionen) Dieses Programm ist für das Produzieren von pdfs ausgelegt. Bei anderen Formaten (z.B. jpeg) treten gerne Bugs mit dem Latex-Compiler auf.

Zeile 13-17: Die Ergebnisse des Fits werden zur Berechnung der Größen τ und U_0, sowie deren Unsicherheiten verwendet.

Zeile 18: Anzeigen der Ergebnisse für τ und U_0

Matlab eignet sich nun ausgezeichnet um schnell und automatisch weitere Rechnungen durchzuführen.

Listing 6.2: Daten in einem ASCII file (linfit.dat). e-4 steht hierbei für $\times 10^{-4}$

```
1 0.000e+00   1.015   0.010
2 5.000e-05   0.946   0.010
3 1.000e-04   0.896   0.010
4 1.500e-04   0.858   0.010
5 2.000e-04   0.825   0.010
6 2.500e-04   0.770   0.010
```

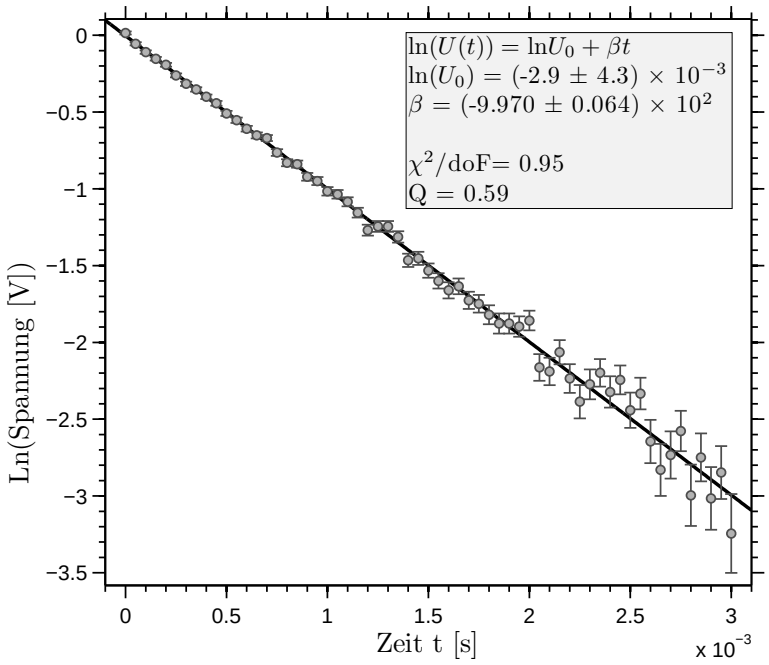

Abbildung 6.1.1: Fit des Logarithmus der Spannung eines Kondensators bei Entladung

7	3.000e−04	0.729	0.010
8	3.500e−04	0.703	0.010
9	4.000e−04	0.670	0.010
10	4.500e−04	0.642	0.010
11	5.000e−04	0.601	0.010
12	5.500e−04	0.575	0.010
13	6.000e−04	0.544	0.010
14	6.500e−04	0.521	0.010
15	7.000e−04	0.512	0.010
16	7.500e−04	0.466	0.010
17	8.000e−04	0.436	0.010
18	8.500e−04	0.432	0.010
19	9.000e−04	0.398	0.010
20	9.500e−04	0.387	0.010

7 Zusätzliche Optionen und Einstellungen

Optionen sind ein nützliches Hilfsmittel um in Matlab optionale Werte an Funktionen zu übergeben. Dazu gibt man zunächst in einem String an, für welche Option der nachfolgende Wert gelten soll. Nach einem Komma folgt dann der eigentliche Wert. Für `wnonlinfit` heißt das z.b.

```
1 wnonlinfit(x,y,yerr,fitfunc,c0,'plot',false)
```

wodurch die Option plot auf `false` gesetzt wird und somit keine Plots ausgegeben werden. Jede Option besitzt intern einen Standardwert, der verwendet wird, wenn nichts spezifiziert wird. Im folgenden werden alle Optionen für wlinfit und wnonlinfit aufgelistet und beschrieben.

Optionen

plot: Variablentyp: Boolean. Legt fest, ob Plots ausgegeben werden. Da plotten bei weitem die meiste Zeit kostet, lohnt es, wenn man keine Plots benötigt die Option auf false zu setzen. Standard: true.

label: Variablentyp: Cell of Strings. Legt die Beschriftungen der Achsen und Benennung der Variablen in der Legende fest. Bsp: 'x-achse' 'y-achse' 'name1' 'name2'. Standard: 'x-axis' 'y-axis' 'c_0' 'c_1' ....

position: Variablentyp: 1×2 Matrix of double. Legt die Position der Legende fest. Dabei wird die Position der linken oberen Ecke angegeben. Standard: kompliziert, hängt von der Funktion und den Achsen ab.

chitol: Variablentyp: double. Gibt die Toleranz für $\frac{\chi^2}{\mathrm{dof}}$ an. Standard: 2 (nur für wnonlinfit).

grid: Variablentyp: Boolean. Legt fest, ob ein Grid in die Plots gelegt werden soll. Standard: false.

print: Variablentyp: Boolean. Legt fest, ob etwas in die Matlab ausgegeben werden soll. Standard: true.

header: Variablentyp: Cell of Strings. Definiert die Überschrift in der Legende. Jedes Element der Cell legt eine Zeile fest. Bsp.: siehe Listing 4.1. Standard: {'(Non)linear Fit'}.

printchi: Variablentyp: Boolean. Legt fest, ob $\frac{\chi^2}{\mathrm{dof}}$ und Q in der Legende aufgeführt werden.

errprec: Variablentyp: integer. Legt fest, wie viele signifikante Stellen in der Legende dargestellt werden sollen. Standard: 2.

axis: Variablentyp: 1×4 Matrix of double. Wie der normale `axis` Befehl. Standard: Kompliziert, hängt von den Daten ab.

contourplot: Variablentyp: Cell of 1×2 matrices. Hiermit können Konturplots der Wahrscheinlichkeitsverteilung der Fitparameter, die in dem Vektor festgelegt werden, erzeugt werden. Daran können Korrelationen zwischen zwei Fitparametern erkannt werden. Standard: keine Ausgabe.

polyfit: Variablentyp: Integer. Fügt den zu fittenden Funktionen ein Polynom mit der angegebenen Ordnung hinzu. Wenn man nur mit einem Polynom fitten möchte, kann man einfach als Cell in der die Funktionen übergeben werden eine leere Cell übergeben und nur die Option `polyfit` verwenden.

Literaturverzeichnis

[1] U. Wolff, P. Fritzsch, and T. Korzec, "Computational Physics I." http://www.physik.hu-berlin.de/com/teachingandseminars/previousCPI.

[2] J. Beringer et al. (Particle Data Group) *Phys. Rev. D86*, p. 010001, 2012.

[3] A. R. Gallant, "Nonlinear regression," *The American Statistician*, vol. 29, no. 2, pp. pp. 73–81, 1975.

[4] U. Müller, "Einführung in die Messung, Auswertung und Darstellung experimenteller Ergebnisse in der Physik." http://gpr.physik.hu-berlin.de/Skripten/Einfuehrung/PDF-Datei/Einfuehrung.pdf.