Mathematisch-Naturwissenschaftliche Fakultät

Institut für Physik

Kurzeinführung in die Messunsicherheitsberechnung für das Physik-Praktikum

Mit Python Code Beispielen

Julien Kluge - 13. Mai 2024 - Version 2.0.1

Inhaltsverzeichnis

1 Mittelwerte

1.1 Arithmetischer Mittelwert

Der *normale*, arithmetische Mittelwert $\overline{x}$ sollte ja noch aus der Schule bekannt sein. Man addiert einfach alle Werte x_i zusammen, und teilt durch die Anzahl n aller Werte.

$$\overline{x} = \frac{x_1 + x_2 + \cdots + x_{n-1} + x_n}{n} \tag{1}$$

Mit Summenschreibweise (die wir als Physiker natürlich bevorzugen weil sie kompakter ist) ergibt sich die Standardmäßige Definition:

Definition 1: Arithmetischer Mittelwert

$$\overline{x} = \frac{1}{n}\sum_{i}^{n} x_i \tag{2}$$

$\overline{x}$	Arithmetischer Mittelwert
n	Anzahl aller (Mess)Werte
x_i	i-ter (Mess)Wert

Python Code 1: Arithmetischer Mittelwert

Wir definieren eine Liste aller (Mess)Werte und berechnen den Mittelwert mit der Funktion **mean** aus dem *statistics* package:

```
from statistics import mean
messwerte = [1.0456, 0.9774, 1.0023, 0.9904, 1.3995]
mean(messwerte)
```

```
1.08304
```

1.2 Gewichteter Mittelwert

Man stelle sich folgende Situation vor: man hat die Werte x_i und möchte aus denen ein Mittelwert bilden. Jetzt weiß man aber, dass einige dieser Werte viel genauer und/oder weniger Unsicher sind als andere. Da hätte man es doch lieber, diese Werte stärker in den Mittelwert eingehen zu lassen. Dafür existiert der gewichtete Mittelwert. In mathematischen Worten also folgendes: man habe die Werte xi mit den Unsicherheiten ui und führe einen Gewichtungsfaktor p_i ein, der von der Unsicherheit abhängt: $p_i = p_i(u_i)$. Jetzt ist uns klar, das wenn die Unsicherheit u_i klein ist, dann wollen wir den Wert stärker gewichten, andersrum wollen wir eine kleine Gewichtung für größere Unsicherheiten. Des weiteren gilt, die statistische Streuung eines Wert mit Unsicherheit u_i ist einfach mit dem Quadrat gegeben, es gilt also:

$$p_i \propto \frac{1}{u_i^2} \tag{3}$$

Unter Einführung einer Proportionalitätskonstante C bekommen wir

$$p_i = \frac{C}{u_i^2} \tag{4}$$

Wenn man diesen Gewichtungsfaktor in die Formel (6) und Formel (7) einsetzt, bemerkt man dass der Proportionalitätsfaktor fast frei gewählt werden darf mit $C \in \mathbb{R}\backslash\{0\}$. Für alle die mit einer Programmiersprache arbeiten, kann der Faktor demnach der Einfachheit halber zu eins gesetzt werden:

Definition 2: Instrumenteller Gewichtungsfaktor

$$p_i = \frac{1}{u_i^2} \tag{5}$$

p_i	Gewichtungsfaktor des Werts $(x_i \pm u_i)$
u_i	Unsicherheit des Werts $(x_i \pm u_i)$

Für alle anderen die per Hand mit Taschenrechner, Excel, o.Ä. rechnen, kann eine Vereinfachung gemacht werden indem man $C = u_1^2$ setzt. Damit wird $p_1 \equiv 1$ und alle anderen Gewichtungsfaktoren sind relative Quotienten zu p_1. Das ist aber natürlich nicht unbedingt notwendig. Jetzt kann das gewichtete Mittel definiert werden mit:

Definition 3: Gewichteter Mittelwert

$$\overline{x} = \frac{\sum_i^n p_i \cdot x_i}{\sum_i^n p_i} \tag{6}$$

$\overline{x}$	Gewichtungsfaktor des Werts $(x_i \pm u_i)$
n	Anzahl aller Werte
x_i	i-ter Wert
p_i	i-ter Gewichtungsfaktor

Natürlich ist auch die Unsicherheit des Mittelwertes damit betroffen:

$$\overline{u} = \pm \frac{\sqrt{\sum_i^n (p_i \cdot u_i)^2}}{\sum_i^n p_i} \tag{7}$$

Durch einsetzen des instrumentellen Gewichtungsfaktors und Vereinfachung erhält man:

Definition 4: Unsicherheit des gewichteten Mittelwerts

$$\overline{u} = \pm \sqrt{\frac{C}{\sum_i^n p_i}} \quad \overset{C=1}{\Longrightarrow} \quad \overline{u} = \pm \left(\sum_i^n p_i \right)^{-1/2} \tag{8}$$

$\overline{u}$	Unsicherheit des gewichteten Mittelwerts
n	Anzahl aller Werte
C	frei wählbarer Proportionalitätsfaktor
p_i	i-ter Gewichtungsfaktor

Beispiel 1: Gewichteter Mittelwert

Folgende Werte der Erdbeschleunigung g mit Unsicherheit u wurden gemessen:

$g\,[\mathrm{m/s^2}]$	9.81	9.79	9.80	9.60
$\pm u\,[\mathrm{m/s^2}]$	9.03	9.11	9.04	9.70

Mit $C = 1$ berechnet sich:

	$p = 1/u^2$	$p \cdot g$
	1111.11	10900
	82.6446	809.091
	625	6125
	2.04082	19.5918
Σ	1820.8	17853.7

Es folgt das Ergebnis mit:

$$\overline{g} = \frac{\sum_i^n p_i \cdot g_i}{\sum_i^n p_i} = \frac{17853.7}{1820.8} = 9.80542 \tag{9}$$

$$\overline{u} = \pm\frac{1}{\sqrt{\sum_i^n p_i}} = \pm\frac{1}{\sqrt{1820.8}} = \pm 0.02344 \tag{10}$$

Und damit:

$$g = (9.81 \pm 0.03)\,\mathrm{m/s^2} \tag{11}$$

Python Code 2: Gewichteter Mittelwert

Wir können die Definition direkt anwenden (**sqrt**: Quadratwurzel, **sum**: Summe einer Liste):

```
from math import sqrt
g = [9.81, 9.79, 9.80, 9.60]
u = [0.03, 0.11, 0.04, 0.70]
p = [1.0 / ui**2 for ui in u]
gp = [gi * pi for gi, pi in zip(g, p)]
[sum(gp) / sum(p), 1.0 / sqrt(sum(p))]
```

```
[9.805424275180432, 0.023435233683447708]
```

Alternativ kann man das geschickter über numpy lösen:

```
import numpy as np
g = np.array([9.81, 9.79, 9.80, 9.60])
u = np.array([0.03, 0.11, 0.04, 0.70])
p = 1.0 / u**2
[sum(g * p) / sum(p), 1.0 / np.sqrt(sum(p))]
```

```
[9.805424275180432, 0.023435233683447708]
```

Wichtige Anmerkung 1

Das gewichtete Mittel darf nur in Fällen gebildet werden in denen sicher ist, dass die Werte normalverteilt sind. Im Allgemeinen kann man sagen: bei zuvielen Ausreißern sollte man auf das normale arithmetische Mittel zurückgreifen.

2 Phytagoreische Addition

Wenn man mehre Unsicherheiten für einen Wert zusammenführen will, muss das geometrisch geschehen. Dafür benutzt man die phytagoreische Addition. Sie ist folgendermaßen definiert:

$$u = \sqrt{u_1^2 + u_2^2 + \cdots + u_{n-1}^2 + u_n^2} \tag{12}$$

Mit Summenschreibweise führt das auf:

Definition 5: Phytagoreische Addition

$$u = \sqrt{\sum_{i}^{n} x_i^2} \tag{13}$$

u	Phytagoreisch addierter Wert
n	Anzahl aller Werte
u_i	i-ter Wert

3 Standardabweichung

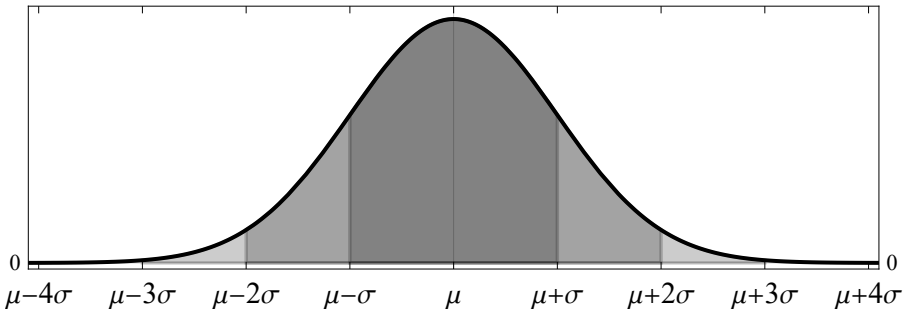


Abbildung 1: Normalverteilung mit Mittelwert μ und Standardabweichung σ.

Ist ein Messwert mit zufälligen Messunsicherheiten belegt, so streuen diese um einen gemeinsamen Punkt (ohne Systematische Abweichungen ist dieser Punkt der Wahrheitswert). Dabei nimmt diese Streuung die Form einer Gaußschen Normalverteilung $N(x, \mu, \sigma)$ (auch Glockenkurve genannt) an. Die Form einer Glocke erklärt sich leicht, wenn man annimmt, dass Streuungen mit steigendem Abstand vom Mittelwert μ zunehmend unwahrscheinlicher werden. Die Wahrscheinlichkeit selbst ist die Fläche unter der Kurve. Es gilt also:

$$\int_{-\infty}^{\infty} \mathrm{d}x\, N(x, \mu, \sigma) = 1 \tag{14}$$

Man sieht außerdem leicht, dass diese Verteilung symmetrisch um den Mittelwert μ ist. Damit können nun symmetrische Intervalle festgelegt werden in denen eine bekannte Wahrscheinlichkeit existiert, dass eine Messung x' dort landet. Diese Intervalle nennt man Standardabweichung σ und sind selbst Parameter der Normalverteilung. Für die ersten fünf Intervalle gilt:

Intervall	statistische Sicherheit
$\mu \pm 1\sigma$	$\approx 68.26895\%$
$\mu \pm 2\sigma$	$\approx 95.44997\%$
$\mu \pm 3\sigma$	$\approx 99.73002\%$
$\mu \pm 4\sigma$	$\approx 99.99367\%$
$\mu \pm 5\sigma$	$\approx 99.99994\%$

Wir rechnen in der Physik für gewöhnlich mit einer statistischen Unsicherheit von einer Standardabweichung (ein Sigma). Hat man eine Messreihe gemacht, kann die Standardabweichung davon berechnet werden mit:

Definition 6: Empirische Standardabweichung

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{i}^{n}(x_i - \mu)^2} \tag{15}$$

σ	Standardabweichung (auch häufig s)
n	Anzahl aller Messwerte
x_i	i-ter Messwert
μ	Arithmetischer Mittelwert $\overline{x}$ der Messwerte

Beispiel 2: Standardabweichung

Folgende zehn Messwerte ($n = 10$) für die Erdbeschleunigung $g\,[\mathrm{m/s^2}]$ wurden aufgenommen:

9.81279, 9.83616, 9.76557, 9.78496, 9.84230
9.75096, 9.72767, 9.84866, 9.74724, 9.76847

Der Mittelwert ist

$$\overline{g} = \mu = 9.788478 \tag{16}$$

Wir berechnen

	$(g-\mu)^2$
	0.0005910733
	0.0022735731
	0.0005247765
	0.0000123763
	0.0028968077
	0.0014076003
	0.0036976129
	0.0036218731
	0.0017005726
	0.0004003201
Σ	0.0171265860

Und somit ergibt sich die Standardabweichung zu:

$$\sigma = \sqrt{\frac{1}{10-1}\cdot 0.0171265860} \tag{17}$$

$$\approx 0.0436228610 \tag{18}$$

Python Code 3: Standardabweichung

Python kennt eine fertige Funktion **stdev** für die Standardabweichung vom *statistics* package:

```
from statistics import stdev
g = [9.81279, 9.83616, 9.76557, 9.78496, 9.84230,
     9.75096, 9.72767, 9.84866, 9.74724, 9.76847]
stdev(g)
```

```
0.04362286092813679
```

ACHTUNG: numpy hat zwar auch eine Funktion für die Standardabweichung aber diese benutzt $\frac{1}{n}$ statt $\frac{1}{n-1}$. Das hat Gründe aber ist generell NICHT das was wir wollen! Man bekommt das aber zurück über das *ddof* argument:

```
import numpy as np
g = [9.81279, 9.83616, 9.76557, 9.78496, 9.84230,
     9.75096, 9.72767, 9.84866, 9.74724, 9.76847]
np.std(g, ddof=1)
```

```
0.04362286092813679
```

4 Vertrauensbereich / zufällige Messunsicherheit

Mithilfe der Standartabweichung lässt sich, in einer bestimmten statistischen Wahrscheinlichkeit, die zufällige Messunsicherheit abschätzen. Diese Abschätzung nennt sich Vertrauensbereich und ist folgendermaßen definiert:

Definition 7: Vertrauensbereich

$$\overline{s} = \pm t \cdot \frac{\sigma}{\sqrt{n}} \tag{19}$$

$\overline{s}$	Vertrauensbereich
σ	Standardabweichung
n	Anzahl aller Messwerte
t	Student T-Wert

4.1 Student T-Wert

Der Student-Faktor (benannt nach William Sealy Gosset der unter dem Pseudonym Student gearbeitet hat) kann auf komplizierte Weise, von der Student T-Verteilung berechnet werden.

Diese Verteilung besitzt nur die Anzahl der Messwerte n (eigentlich Freiheitsgrade: $\nu = n - 1$) als Parameter und ist um null symmetrisch. In den Einführungspraktika und Grundpraktikum I & II ist es erlaubt die Näherung $t \approx 1$ für $n \geq 6$ zu machen. Will man diese Näherung nicht eingehen, kann man in folgende Tabelle gucken und den Wert ablesen.

n	$\pm 1\sigma$	$\pm 2\sigma$	$\pm 3\sigma$	$\pm 4\sigma$	$\pm 5\sigma$
2	1.83734	13.9677	235.801	10050.4	1.1×10^6
3	1.32128	4.52654	19.2067	125.641	1320.71
4	1.19688	3.30682	9.21894	32.6164	156.678
5	1.14163	2.86931	6.62021	17.4482	56.8484
6	**1.11051**	2.64865	5.50708	12.2814	31.8470
7	1.09057	2.51652	4.90406	9.84416	22.0199
8	1.07671	2.42881	4.52997	8.46693	17.1025
9	1.06653	2.36642	4.27663	7.59506	14.2495
10	1.05873	2.31981	4.09426	6.99864	12.4220
11	1.05256	2.28368	3.95694	6.56718	11.1662
12	1.04757	2.25486	3.84994	6.24163	10.2571
13	1.04344	2.23135	3.76428	5.98780	9.57203
14	1.03997	2.21180	3.69419	5.78466	9.03909
15	1.03701	2.19529	3.63580	5.61858	8.61377
16	1.03446	2.18116	3.58642	5.48038	8.26710
17	1.03224	2.16894	3.54413	5.36364	7.97950
18	1.03029	2.15826	3.50750	5.26377	7.73733
19	1.02856	2.14885	3.47547	5.17739	7.53077
20	1.02702	2.14049	3.44723	5.10196	7.35262
21	1.02563	2.13303	3.42215	5.03554	7.19748
22	1.02438	2.12631	3.39973	4.97661	7.06121
23	1.02325	2.12024	3.37956	4.92398	6.94062
24	1.02222	2.11473	3.36132	4.87670	6.83316
25	1.02127	2.10970	3.34475	4.83400	6.73684
26	1.02040	2.10509	3.32963	4.79524	6.65001
27	1.01960	2.10085	3.31578	4.75990	6.57136
28	1.01886	2.09694	3.30305	4.72756	6.49979
29	1.01818	2.09333	3.29130	4.69785	6.43440
30	1.01754	2.08997	3.28043	4.67045	6.37442
31	1.01695	2.08684	3.27033	4.64512	6.31921
32	1.01639	2.08393	3.26094	4.62163	6.26824
33	1.01587	2.08120	3.25218	4.59978	6.22102
34	1.01538	2.07865	3.24399	4.57942	6.17717
35	1.01492	2.07625	3.23631	4.56038	6.13635
36	1.01449	2.07400	3.22910	4.54255	6.09824
37	1.01408	2.07187	3.22231	4.52582	6.06258
38	1.01370	2.06986	3.21592	4.51009	6.02916
39	1.01333	2.06796	3.20988	4.49527	5.99777
40	1.01299	2.06617	3.20417	4.48128	5.96822
50	1.01031	2.05232	3.16051	4.37526	5.74678
100	1.00508	2.02557	3.07755	4.17845	5.34785
1000	1.00050	2.00251	3.00752	4.01708	5.03272

Wir sehen, dass die im Praktikum verwendete Näherung mit $n \geq 6$ willkürlich und den Ver-

suchen zweckmäßig gewählt worden ist. Der Student-T Wert ist $t(n = 6) \approx 1.11051$ d.h. bei $n = 6$ unterschätzen wir den Vertrauensbereich/die zufällige Messunsicherheit um circa 11%.

4.2 Größtfehlerabschätzung

Es stellt sich die Frage, was man benutzt wenn $n < 6$ und die Näherung des Student T-Wertes nicht mehr gemacht werden kann. In diesem Fall kann man eine Größtfehlerabschätzung machen. In einfachen Worten nimmt man die Größtmöglichen Abweichungen der Messreihe an, indem man sich anguckt welcher Wert am weitesten vom Mittelwert entfernt ist und nimmt das als Unsicherheit. Es gilt also:

Definition 8: Größtfehlerabschätzung als Vertrauensbereich

$$\overline{s} \approx \pm \max_i |x_i - \overline{x}| \tag{20}$$

$\overline{s}$	Abgeschätzter Vertrauensbereich
x_i	i-ter Messwert
$\overline{x}$	Mittelwert

Beispiel 3: Vertrauensbereich mit Standartabweichung

Wir nehmen die Werte des vorherigen Beispiels:

$$\sigma \approx 0.04362$$
$$n = 10$$

Es ergibt sich **ohne Student T-Wert**:

$$\overline{s} = \pm \frac{0.04362}{\sqrt{10}} \approx \pm 0.01379 \tag{21}$$

Mit Student T-Wert ($t(n = 10) \approx 1.05873$):

$$\overline{s} = \pm 1.05873 \cdot \frac{0.04362}{\sqrt{10}} \approx \pm 0.01460 \tag{22}$$

Mit **Größtfehlerabschätzung** (auch wenn sie hier wirklich nutzlos ist):

$$\overline{s} = \pm \max \left(0.0243, 0.0477, 0.0229, 0.0035, 0.0538, 0.0375, 0.0608, 0.0602, 0.0412, 0.0200\right)$$
$$\overline{s} = \pm 0.0608 \tag{23}$$

Python Code 4: Vertrauensbereich

Python kann ohne den Student-T Wert rechnen. Um den Student-T Wert zu Berechnen, müssen externe Packages eingebunden werden. Das machen wir hier nicht:

```
import numpy as np
g = [9.81279, 9.83616, 9.76557, 9.78496, 9.84230,
     9.75096, 9.72767, 9.84866, 9.74724, 9.76847]
[
       np.std(g, ddof=1) / np.sqrt(len(g)),
       max(abs(g - np.mean(g)))
]
```

```
[0.013794759858567901, 0.06080800000000153]
```

5 Runden und signifikante Stellen

Ergebnisse auf 20 Stellen genau zu berechnen ist zwar eine tolle Leistung aber nicht sehr sinnvoll. Es macht keinen Sinn, Werte jenseits von ihren Unsicherheitsgrenzen genau anzugeben, deswegen müssen wir auf richtige Stellen runden. Im Allgemeinen runden wir so, dass die erste Stelle der Unsicherheit ungleich null, gleichzeitig die letzte Ziffer von rechts ist. Dabei wird der Messwert kaufmännisch gerundet, während die Unsicherheit stets aufgerundet wird. Dabei müssen nachgehende Nullen vom Wert zwingend mitgeschrieben werden. Die so noch erhaltenen Stellen werden als signifikant bezeichnet.

Beispiel 4: Runden

$$(6.3279 \pm 0.057) \longrightarrow (6.33 \pm 0.06) \tag{24}$$
$$(36.03 \pm 0.41) \longrightarrow (36.0 \pm 0.5) \tag{25}$$

5.1 DIN

Wie für alles, existiert eine eigene DIN-Norm mit einer Eigenheit. Es gilt, wenn die Unsicherheit mit der ersten Stelle auf eins oder zwei geht, dann muss die Signifikanz auf eine weitere Stelle ausgedehnt werden. Nachgehende Nullen die hierbei durch Runden entstehen können, müssen dieses mal auch bei der Unsicherheit mitgenommen werden.

Beispiel 5: Runden nach DIN

$$(6.3279 \pm 0.134) \longrightarrow (6.33 \pm 0.14) \tag{26}$$
$$(36.003 \pm 0.148) \longrightarrow (36.00 \pm 0.15) \tag{27}$$
$$(15.437 \pm 0.297) \longrightarrow (15.44 \pm 0.30) \tag{28}$$

5.2 Angabe von Naturkonstanten

Bei der Angabe von Naturkonstanten kann man teilweise eine andere Notation von Unsicherheiten beobachten. Hier wird hinter dem Wert die Unsicherheit eingeklammert welche mit signifikanter Stellenzahl von rechts gesehen geht.

Beispiel 6: Notation von Naturkonstanten

Das plancksche, reduzierte Wirkungsquantum wurde vor der Neudefinition 2019 definiert mit:

$$\hbar = 1.054571800(13) \times 10^{-34}\,\mathrm{J\,s} \tag{29}$$
$$= (1.054571800 \pm 0.000000013) \times 10^{-34}\,\mathrm{J\,s} \tag{30}$$

Der Vorteil dieser Notation ist offensichtlich. Einige Beispiele:

$$(6.33 \pm 0.06) \longrightarrow 6.33(6) \tag{31}$$
$$(36.0 \pm 2.5) \longrightarrow 36.0(2.5) \tag{32}$$
$$(6.33 \pm 0.14) \longrightarrow 6.33(14) \tag{33}$$
$$(15.44 \pm 0.30) \longrightarrow 15.44(30) \tag{34}$$

6 Kochrezept: Messwert berechnen

Hat man nun sukzessiv die erste Messreihe für einen Wert durchgeführt, gibt es ein Kochrezept wie wir diesen sowie seine Unsicherheit ordentlich berechnen können:

Definition 9: Messwertberechnung

1. Mittelwert berechnen

$$\overline{x} = \frac{1}{n}\sum_{i}^{n} x_i \tag{35}$$

2. Bekannte systematische Fehler korrigieren (sehr selten)

$$\overline{x}_k = \overline{x} - u_{\text{sys}} \tag{36}$$

3. Vertrauensbereich berechnen

$$\overline{s} = \pm \begin{cases} \frac{\sigma}{\sqrt{n}} = \sqrt{\frac{1}{n(n-1)}\sum_i^n (x_i - \overline{x})^2} & n \geq 6 \\ \max\limits_i |x_i - \overline{x}| & n < 6 \end{cases} \tag{37}$$

4. Messunsicherheit durch phytagoreisches Addieren aller Unsicherheiten berechnen

$$u = \sqrt{\overline{s}^2 + u_{\text{gerät}}^2 + u_{\text{ablese}}^2 + \dots} \tag{38}$$

5. Ergebnis auf signifikante Stellen runden, mit Einheit (und Zehnerpotenz oder SI-Präfix) angeben.

$$(\overline{x}_k \pm u) \times 10^m \,[\text{SI-Präfix Einheit}] \tag{39}$$

Wichtige Anmerkung 2

Bei Angabe von Einheiten muss stets darauf geachtet werden, dass diese **nicht kursiv** geschrieben sind um sie nicht mit Variablen zu verwechseln.

7 Messunsicherheitsfortpflanzung/Fehlerfortpflanzung

Wenn man eine Funktion $y = f(a, b, c, \dots)$ mit den Werten $a, b, c, \dots$ und den dazugehörigen Unsicherheiten $u_a, u_b, u_c, \dots$ hat, kann man leicht y ausrechnen. Doch wie rechnet man die Unsicherheit u_y für y aus? Dafür existiert die Gaußsche Messunsicherheitsfortpflanzung.

7.1 Unkorrelierte Fortpflanzung

Wenn die einzelnen Werte x_i mit den Unsicherheiten u_i nicht miteinander korreliert sind und die Funktion $f(x_1, x_2, \dots, x_n)$ ausgerechnet werden soll, dann kann man die Gaußsche Messunsicherheitsfortpflanzung darstellen mit:

Definition 10: Unkorrelierte Gaußsche Messunsicherheitsfortpflanzung

$$u_f = \sqrt{\sum_i^n \left(\frac{\partial f(x_1, x_2, \ldots, x_n)}{\partial x_i} \cdot u_i \right)^2} \tag{40}$$

u_f	Fortgepflanzte Unsicherheit der berechneten Funktion $f(x_1, x_2, \ldots, x_n)$
n	Anzahl aller Werte
x_i	i-te Variable/Wert
u_i	i-te Unsicherheit

Man kann sich diese Formel sogar ganz anschaulich darstellen. Die partielle Ableitung $\partial f / \partial x_i$ am Punkt $(x_1, x_2, \ldots, x_n)$ zeigt den Anstieg nach x_i und damit, wie empfindlich die Formel auf Änderung dieser Variable reagiert. Es fungiert somit als Gewichtung zur Unsicherheit u_i welche Schlussendlich phytagoreisch mit allen anderen addiert wird.

Beispiel 7: Unkorrelierte Messunsicherheitsfortpflanzung

Nach Ohmschen Gesetz gilt folgender Zusammenhang zwischen Spannung U, Stromstärke I und elektrischer Widerstand R:

$$R = \frac{U}{I} \tag{41}$$

Man nehme jetzt folgende Werte (ungerundet) an:

$$U = (238.46 \pm 7.34)\,\mathrm{V} \tag{42}$$
$$I = (0.9239 \pm 0.0081)\,\mathrm{A} \tag{43}$$

Durch schnelle Rechnung ergibt sich:

$$R \approx 258.102\,\Omega \tag{44}$$

Die Unsicherheit u_R von R bestimmt sich unter Vernachlässigung der Korrelation durch Messunsicherheitsfortpflanzung so:

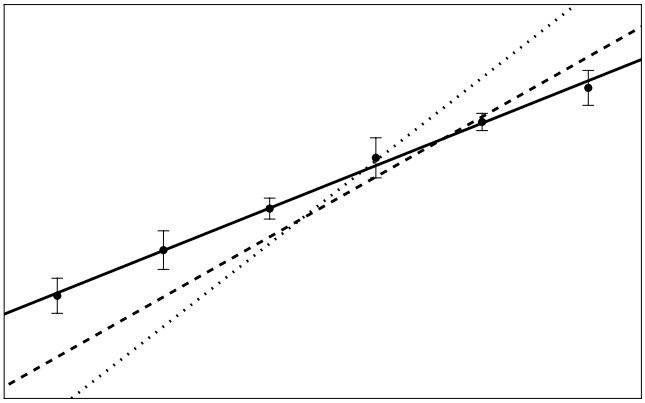
$$u_R = \sqrt{\left(\frac{\partial R}{\partial U} \cdot u_U \right)^2 + \left(\frac{\partial R}{\partial I} \cdot u_I \right)^2} \tag{45}$$
$$= \sqrt{\left(\frac{1}{I} \cdot u_U \right)^2 + \left(-\frac{U}{I^2} \cdot u_I \right)^2} \tag{46}$$
$$= \sqrt{\left(\frac{1}{0.9239} \cdot 7.34 \right)^2 + \left(\frac{238.46}{(0.9239)^2} \cdot 0.0081 \right)^2} \tag{47}$$
$$u_R \approx 8.26055 \tag{48}$$

Damit ist das Ergebnis (gerundet):

$$R = (258 \pm 9)\,\Omega \tag{49}$$

7.2 Exkurs: Korrelation

Ohne viel Hintergrundwissen gilt allgemein: zwei Größen sind miteinander Korreliert, wenn man die eine nicht ohne die andere wählen kann. Man kann sich dieses Prinzip sehr leicht an einer linearen Funktion erklären: In der nebenstehenden Abbildung wurde eine lineare Funktion $f(x) = a \cdot x + b$ möglichst passend durch die vorhanden Punkte gelegt (durchgezogene Linie; wir werden das im nächsten Kapitel unter dem Namen Regression kennenlernen). Wenn man sich nun vorstellt, dass man den Anstieg a dieser Gerade nach oben zwingt (gepunktete Linie), dann muss der Achsenschnittpunkt b sinken damit die Gerade wieder relativ gut in den Punkten liegt (gestrichelte Linie). Das gleiche gilt auch andersherum, zwingt man den Anstieg nach unten, steigt der Achsenabschnitt. Natürlich gilt das auch in vertauschten Rollen, also zwingt man den Achsenabschnitt zu sinken, muss der Anstieg steigen und andersrum. Man merkt damit, dass man die beiden Parameter der linearen Funktion nicht unabhängig voneinander wählen kann. Sie sind also korreliert. Dieses Konzept wird für Messunsicherheitsfortpflanzung wichtig da man für jede Korrelation einen weiteren Term bekommt der berechnet werden muss.



Um diese Korrelation zwischen zwei Größen x_i und x_j zu quantifizieren führt man eine Größe ein die man ganz einfach Korrelationskoeffizient $C_{i,j}$ nennt. Dieser kann Werte von $+1$ (volle Korrelation, die Größen ändern sich miteinander in eine Richtung), bis -1 (vollständige Antikorrelation, die Größen ändern sich gegeneinander in die entgegengesetzte Richtung) annehmen. Ein Wert von $C_{i,j} = 0$ bedeutet die Größen sind unkorreliert und können sich unabhängig voneinander ändern. Zugehörig zum Korrelationskoeffizient $C_{i,j}$ gibt es die Kovarianz $u_{i,j}$. Diese Kovarianz ist in gewisser Weise sehr ähnlich zur Varianz u_i^2 einer einzigen Variable. Alle diese Größen kann man auch zusammen in einer Formel darstellen:

Definition 11: Zusammenhang Korrelation, Kovarianz und Unsicherheit

$$C_{i,j} = \frac{u_{i,j}}{u_i \cdot u_j} \tag{50}$$

$C_{i,j}$	Korrelationskoeffizient $C_{i,j} \in [-1, +1]$ der Werte x_i und x_j
$u_{i,j}$	Kovarianz der Werte x_i und x_j
u_i	i-te Unsicherheit des x_i Werts
u_j	j-te Unsicherheit des x_j Werts

Einige nützliche Eigenschaften lassen sich aus dieser Formel ablesen. Zum einen, sind die Korrelationsgrößen immer symmetrisch, d.h. $C_{i,j} = C_{j,i}$ sowie $u_{i,j} = u_{j,i}$. Desweiteren kann die Kovarianz im Betrag nie größer werden als die Multiplikation der dazugehörigen Unsicherheiten $|u_{i,j}| \leq u_i \cdot u_j$.

Jetzt stellt sich natürlich die Frage, wie berechnen wir denn die Korrelation oder Kovarianz? Eine Möglichkeit besteht, wenn man beide Größen gemessen hat. Wie auch mit der empiri-

schen Standardabweichung gibt es eine empirische Formel für den Korrelationskoeffizient. Der sogenannte Pearson Korrelationskoeffizient:

$$C_{i,j} = \frac{1}{(n-1) \cdot \sigma_i \cdot \sigma_j} \sum_{k}^{n} \left(x_{i,k} - \overline{x}_i\right) \left(x_{j,k} - \overline{x}_j\right) \tag{51}$$

Wobei σ_i die Standardabweichung von der Messreihe x_i ist, $x_{i,k}$ das k-te Element von x_i und $\overline{x}_i$ der Mittelwert der Messreihe. Man kann diesen auch als reine Summe darstellen:

$$C_{i,j} = \frac{\sum_{k}^{n} \left(x_{i,k} - \overline{x}_i\right) \left(x_{j,k} - \overline{x}_j\right)}{\sqrt{\sum_{k}^{n} \left(x_{i,k} - \overline{x}_i\right)^2} \cdot \sqrt{\sum_{k}^{n} \left(x_{j,k} - \overline{x}_j\right)^2}} \tag{52}$$

Hat man diesen Wert ausgerechnet, kann man leicht über die Unsicherheiten dann die Kovarianz abschätzen.
Eine zweite Methode die Kovarianzen und Korrelationen abzuschätzen werden wir später im Kapitel zu Regressionen noch kennenlernen.

Python Code 5: Pearson Korrelationskoeffizient

Python kennt die Funktion **correlation** für den Korrelationskoeffizient im *statistics* Package:

```
from statistics import correlation
x1 = [-0.92, -0.57, -0.93, -0.82, -0.45, 0.76, 0.78, 0.93, 0.37, 0.48]
x2 = [-2.61, -2.01, -2.58, -2.46, -1.91, 0.45, 0.37, 0.6, -0.51, 0.03]
correlation(x1, x2)
```

```
0.9973191674611046
```

7.3 Korrelierte Fortpflanzung

Für jede Korrelation der beiden Variablen x_i und x_j mit der zugehörigen Kovarianz $u_{i,j}$, addiert sich ein weiterer Term der Form

$$2 \cdot \frac{\partial f(x_1, x_2, \ldots, x_n)}{\partial x_i} \cdot \frac{\partial f(x_1, x_2, \ldots, x_n)}{\partial x_j} \cdot u_{i,j} \tag{53}$$

mit unter die Wurzel der unkorrelierten Fortpflanzung. Man kann also allgemein schreiben:

Definition 12: Gaußsche Messunsicherheitsfortpflanzung

$$u_f = \sqrt{\sum_i^n \left(\frac{\partial f(x_1,\dots)}{\partial x_i} \cdot u_i\right)^2 + 2\sum_i^{n-1}\sum_{j=i+1}^{n} \frac{\partial f(x_1,\dots)}{\partial x_i} \cdot \frac{\partial f(x_1,\dots)}{\partial x_j} \cdot u_{i,j}} \tag{54}$$

u_f	Fortgepflanzte Unsicherheit der berechneten Funktion $f(x_1, x_2, \dots, x_n)$
n	Anzahl aller Werte
x_i	i-te Variable/Wert
u_i	i-te Unsicherheit
$u_{i,j}$	Kovarianz der Werte x_i und x_j

Beispiel 8: Korrelierte Messunsicherheitsfortpflanzung

Gleiches Beispiel wie zuvor im unkorrelierten Beispiel mit den selben Werten und einem Kovarianzfaktor von

$$u_{U,I} = -0.109 \tag{55}$$

Es folgt mit Korrelationsterm:

$$u_R = \sqrt{\left(\frac{\partial R}{\partial U} \cdot u_U\right)^2 + \left(\frac{\partial R}{\partial I} \cdot u_I\right)^2 + 2 \cdot \frac{\partial R}{\partial U} \cdot \frac{\partial R}{\partial I} \cdot u_{U,I}} \tag{56}$$

$$= \sqrt{\left(\frac{1}{I} \cdot u_U\right)^2 + \left(-\frac{U}{I^2} \cdot u_I\right)^2 - 2 \cdot \frac{U}{I^3} \cdot u_{U,I}} \tag{57}$$

$$= \sqrt{\left(\frac{1}{0.9239} \cdot 7.34\right)^2 + \left(\frac{238.46}{(0.9239)^2} \cdot 0.0081\right)^2 - 2\frac{238.46}{(0.9239)^3}(-0.109)} \tag{58}$$

$$u_R \approx 11.5925 \tag{59}$$

Und somit das Ergebnis (gerundet):

$$R = (258 \pm 12)\,\Omega \tag{60}$$

Wichtige Anmerkung 3

Korrelationsterme können (anders als unkorrelierte) die Unsicherheit sowohl in positive als auch negative Richtung verändern.

Wenn man sich die Formel für die korrelierte Messunsicherheitsfortpflanzung so ansieht könnte man darauf kommen, dass dahinter eine Matrix-Vektor Multiplikation steht. Tatsächlich, wenn man eine Kovarianzmatrix $\underline{U}$ definiert in der Form $(\underline{U})_{i,j} = u_{i,j}$ dann kann man schreiben:

$$u_f = \sqrt{\nabla^T f \cdot \underline{U} \cdot \nabla f} \tag{61}$$

Diese Matrix $\underline{U}$ wird uns im Kapitel über Regressionen begegnen.

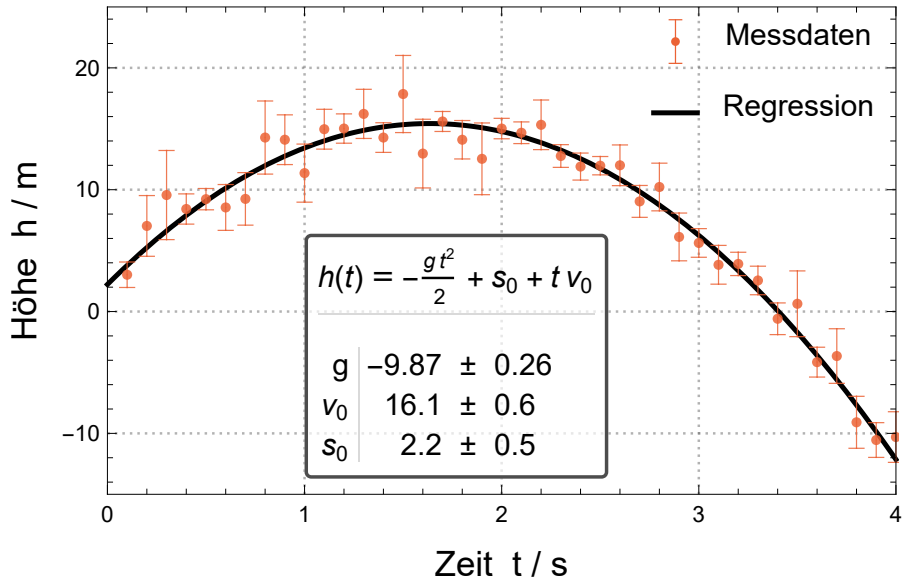


Abbildung 2: Messung und Regression einer ballistischen Flugkurve $h(t) = -g/2 \cdot t^2 + v_0 \cdot t + s_0$. Die Parameter g, v_0, s_0 wurden durch Regression mit Unsicherheiten aus den Daten bestimmt.

8 Regressionen

Es gibt viele Wege wie man die Regression (auch: Ausgleichsrechnung oder Fit) fachlich und mathematisch richtig einführen kann. Die einfachste Form ist aber zu sagen: Wir haben n viele Wertepaare (x_i, y_i) mit den dazugehörigen Unsicherheiten $(u_{x,i}, u_{y,i})$ und eine Gleichung $y = f(x, \{\alpha, \beta, \gamma, \dots\})$ die den Verlauf dieser Punkte beschreiben sollte. Wie müssen wir jetzt die Parameter α, β, $\gamma, \dots$ dieser Gleichung wählen, dass sie am besten zu unseren Daten passt? Mathematisch basiert das auf einem Verfahren das den Abstand zwischen Funktionswerten $y = f(x_i)$ zu gemessenen Werten y_i minimiert.

$$\{\alpha, \beta, \gamma, \dots\} = \min_{\alpha,\beta,\gamma,\dots} \left[\sum_i^n \left(y_i - f(x_i, \{\alpha, \beta, \gamma, \dots\}) \right)^2 \right] \tag{62}$$

Dieses Verfahren ist (besonders für nichtlineare Fits) recht umfangreich und soll kein Gegenstand dieser Einführung hier werden. Für gewöhnlich übernehmen Programme/Programmiersprachen diese Arbeit für euch und die Ergebnisse von Regressionen werden dann visuell dargestellt:

8.1 Fit-Modell und Linearität

Wenn es um Regressionen geht, gibt es grundlegend zwei verschiedene Typen: lineare Regressionen und nichtlineare Regressionen. Eine lineare Regression muss nicht zwingend eine lineare Funktion erzeugen. Eine Regression ist dann linear, wenn die Funktion nach der sie gemacht wird ausgedrückt werden kann als die alleinigen Regressionsparameter α, β, $\gamma, \dots$ welche linear kombiniert werden mit linear unabhängigen Basisfunktionen $f_1(x)$, $f_2(x)$, $f_3(x), \dots$.

$$y = f(x) = \alpha \cdot f_1(x) + \beta \cdot f_2(x) + \gamma \cdot f_3(x) + \dots = \sum_i p_i \cdot f_i(x) \tag{63}$$

Beispiel 9: Unterschied nicht/-lineare Regressionen

Funktion	Linear/Nichtlinear
$f(x) = a \cdot x + b$	Linear
$f(x) = a^2 \cdot x + b$	Nichtlinear weil a nichtlinear
$f(x) = a \cdot e^x + b \cdot \log x$	Linear weil Linearkombination aus a, b und $e^x, \log x$
$f(x) = \sin(a \cdot x)$	Nichtlinear weil keine Linearkombination
$f(x) = a \cdot x + 2 \cdot b \cdot x$	Nichtlinear weil Basisfunktionen $x, 2x$ nicht unabhängig
$f(x) = a \cdot e^{-x} + b \cdot x + b \cdot \pi$	Nichtlinear weil b doppelt vorhanden
$f(x) = a \cdot e^{-x} + b \cdot (x + \pi)$	Linear. Linearisiertes Beispiel von Zeile davor
$f(x) = a(x+1)^2 + b(x+2)^2$	Linear weil $(x+1)^2$ und $(x+2)^2$ linear unabhängig
$f(x) = a(x+1) + b \cdot x + c$	Nichtlinear da $(x+1)$ und $x, 1$ linear abhängig

Wichtige Anmerkung 4

Ein Regressionsmodell kann auch über mehrere Basisfunktionen nichtlinear werden. Beispielsweise war das Modell $f(x) = a(x+1)^2 + b(x+2)^2$ linear da die Basisfunktionen $(x+1)^2$ und $(x+2)^2$ linear unabhängig zueinander sind. Das Modell $f(x) = a(x+1) + b(x+2)$ ist somit ebenfalls linear. Aber das Modell $f(x) = a(x+1) + b(x+2) + c(x+3)$ ist nichtlinear da eine der drei Basisfunktionen immer als Linearkombination der anderen beiden geschrieben werden kann. Sowas kommt fairerweise sehr selten vor.

Python Code 6: Lineare und nichtlineare Regression

Python hat keine eingebauten Funktionen für Regressionen. Wir greifen hierbei auf externe Packages zurück. Im Fall der nichtlinearen Regression auf das *SciPy* Package. Das muss vorher natürlich vom eingebauten Package-Manager runtergeladen werden. Die Funktion **curve_fit** gibt die abgeschätzten Parameter sowie die Kovarianzmatrix zurück. Die Unsicherheiten sind gegeben als die Wurzel aus der Diagonale dieser Matrix.

```
import numpy as np
from scipy.optimize import curve_fit
data_x = [-2.0, -1.5, -1.0, -0.5, 0.0, 0.5, 1.0, 1.5, 2.0]
data_y = [-0.62, 0.35, 0.98, 1.4, 1.7, 1.91, 2.09, 2.24, 2.37]
model = lambda x, a, b, c : a*np.exp(-x)+b*x+c
popt, pcov = curve_fit(model, data_x, data_y)
print(popt)
print(np.sqrt(np.diagonal(pcov)))
```

```
[-0.29753859  0.20723109  1.99394192]
[0.00111616 0.00199756 0.00259653]
```

Ich habe kein vollständig funktionales Package für Python gefunden, in welchen man lineare Regressionen machen könnte. Es existieren Spezialfälle wie z.B. numpys **polyfit()** aber keins welches die selbe Funktionalität bietet wie Mathematicas **LinearModelFit**[] oder Julias **GLM.lm()**. Wenn man das dennoch machen muss, dann hat man natürlich die Wahl, selbst die Designmatrix des Systems aufzustellen und diese über **scipy.optimize.lsq_linear()** zu lösen. Daraufhin müsste man natürlich auch selbst die Kovarianzmatrix berechnen um an die Unsicherheiten zu kommen.

8.2 Gewichtete Regression

Wie auch beim gewichteten Mittel haben wir in der Regression die Option genauere/Präzisere Messpunkte stärker in das Ergebnis einfließen zu lassen. Auch hier gilt die bereits bekannte Instrumentelle Gewichtung, d.h. zu jedem Datenpunkt (x_i, y_i) mit den Unsicherheiten $(u_{x,i}, u_{y,i})$ existiert die Gewichtung p_i mit

$$p_1 = \frac{1}{u_{y,i}^2} \tag{64}$$

Hierbei fällt auf, das nur die Unsicherheit von y_i eine Rolle spielt. Dies macht man mit der Näherung, dass $u_{x,i} \ll u_{y,i}$. Sollte diese Näherung umgekehrt gegeben sein, d.h. $u_{x,i} \gg u_{y,i}$, kann man mit $y = f(x)$ die Umkehrfunktion bilden $x = f^{-1}(y)$ und nach (y_i, x_i) mit den Gewichtungen $p_i = 1/u_{x,i}^2$ fitten.

Wichtige Anmerkung 5

Sollte weder die Bedingung $u_{x,i} \ll u_{y,i}$ noch $u_{x,i} \gg u_{y,i}$ erfüllt sein, dann muss man theoretisch eine sogenannte Orthogonalregression machen. Diese ist aber furchtbar kompliziert und nicht Gegenstand dieses Scripts.

Python Code 7: Gewichtete Regression

Mit den Daten und Modell von zuvor kann man gewichten, indem man seine Unsicherheiten direkt als Argument für die **sigma** Option übergibt:

```
unsicherheiten = [-2.18, -1.28, -0.71, -0.32, 0.1, 0.3, 0.71, 1.28, 2.19]
popt, _ = curve_fit(model, data_x, data_y, sigma=unsicherheiten)
print(popt)
```

```
[-0.3022283   0.19789626  2.00119343]
```

8.3 R^2-Test

Der R^2-Test oder auch Bestimmtheitsmaß genannt, gibt an wie viel Prozent der Abweichung von dem Fitmodell erklärt werden kann über aufgespannte Flächen der Residuen. Es ist definiert mit:

$$R^2 = 1 - \frac{\sum_i^n \left(f(x_i) - y_i\right)^2}{\sum_i^n \left(\overline{y} - y_i\right)^2} \tag{65}$$

und kann Werte von null bis eins annehmen. Dabei gilt: je näher der Wert an eins, desto besser. Übliche/gute Werte im Grundpraktikum kommen auf 0.99 und mehr und dieser Test ist in der Physik auch von geringerer Bedeutung.
Beim normalen R^2-Test gibt es ein Problem. Steigt die Anzahl der Parameter, dann steigt zwangsläufig auch der R^2-Test mit an. Um den entgegen zu wirken, kann ein korrigierter Test angegeben werden mit p als die Anzahl der Parameter im Fitmodell:

$$\overline{R}^2 = R^2 - (1 - R^2) \cdot \frac{p}{n - p - 1} \tag{66}$$

Wichtige Anmerkung 6

Der R^2-Test ist in der Physik von eher geringerer Bedeutung da bei uns die Messungen eher weniger stark streuen und somit übliche Werte von anfang an $R^2 \approx 1$ sind.
Der Test eignet sich auch wirklich nur bei linearen Funktionen als Fitmodell.
Zuletzt muss gesagt werden, dass der Test nicht angibt ob ein Fitmodell gut zu den Daten passt oder konvergiert ist.

Python Code 8: R^2-Test

Python hat keine eingebaute Möglichkeit den R^2-Test zu bestimmen. Man muss diesen also manuell ausrechnen:

```
import numpy as np
popt, _ = curve_fit(model, data_x, data_y)
r2_num = sum([(model(x, *popt) - y)**2 for x, y in zip(data_x, data_y)])
r2_den = sum((np.mean(data_y) - np.array(data_y))**2)
r2 = 1 - r2_num / r2_den
print(r2)
```

```
0.9999900825958234
```

8.4 χ^2/dof-Test

Der χ^2/dof-Test (dof steht für Freiheitsgrade [**d**egrees **o**f **f**reedom]) und kann berechnet werden mit $dof = n - p$ wobei p die Anzahl an Fitparameter angibt) gibt begrenzte Auskunft darüber wie gut der Fit innerhalb der Unsicherheiten an die Daten angepasst ist. Er kann Werte von null bis unendlich annehmen wobei ein Wert von $n/(n-p) \overset{n \gg p}{\approx} 1$ als ein guter Test gilt. Abweichungen von diesem Nominalwert können mehrere Gründe haben:

- $\chi^2/dof \ll 1$: Entweder zu große Unsicherheiten oder relativ zu den Unsicherheiten überdurchschnittlich gute Anpassung des Fits an die Daten.
- $\chi^2/dof \gg 1$: Entweder zu kleine Unsicherheiten oder relativ zu den Unsicherheiten unterdurchschnittlich schlechte Anpassung des Fits an die Daten.

Die tatsächlich, finale Einschätzung unterliegt aber einer leichten Subjektivität. Definiert ist der Test über die eingangs erwähnten Abstandsquadrate der Messpunkte (x_i, y_i) mit der Fitfunktion $y = f(x)$ und den zu den Messpunkten gehörenden Messunsicherheiten $(u_{x,i}, u_{y,i})$:

$$\frac{\chi^2}{dof} = \frac{1}{n-p} \sum_i^n \frac{(f(x_i) - y_i)^2}{u_{y,i}^2} \tag{67}$$

8.5 Kovarianz/- und Korrelationsmatrix

Aus dem Kapitel zur korrelierten Gaußschen Messunsicherheitsfortpflanzung wissen wir, dass wir dazu einen Kovarianzfaktor $u_{i,j}$ brauchen. Abgesehen von einer Abschätzung per Hand durch $u_{i,j} = C_{i,j} \cdot u_i \cdot u_j$ oder Messung und Berechnung von Formel (**??**), können diese Werte aus einer Regression in Form einer Kovarianzmatrix gewonnen werden. Diese hat auf den Diagonalen die Quadrate der Unsicherheiten der Parameter und außerhalb die jeweiligen, gesuchten Kovarianzen. Dazu parallel, kann man natürlich auch eine Korrelationsmatrix angeben

Definition 13: Kovarianzmatrix und Korrelationsmatrix

$$\operatorname{cov}(\alpha,\,\beta,\,\gamma,\dots) = \begin{pmatrix} u_\alpha^2 & u_{\alpha,\beta} & u_{\alpha,\gamma} & \dots \\ u_{\beta,\alpha} & u_\beta^2 & u_{\beta,\gamma} & \dots \\ u_{\gamma,\alpha} & u_{\gamma,\beta} & u_\gamma^2 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \tag{68}$$

$$\operatorname{corr}(\alpha,\,\beta,\,\gamma,\dots) = \begin{pmatrix} 1 & C_{\alpha,\beta} & C_{\alpha,\gamma} & \dots \\ C_{\beta,\alpha} & 1 & C_{\beta,\gamma} & \dots \\ C_{\gamma,\alpha} & C_{\gamma,\beta} & 1 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \tag{69}$$

$\alpha,\,\beta,\,\gamma,\dots$	Fitparameter
$\operatorname{cov}(\alpha,\,\beta,\,\gamma,\dots)$	Kovarianzmatrix
$\operatorname{corr}(\alpha,\,\beta,\,\gamma,\dots)$	Korrelationsmatrix
$u_\alpha,\,u_\beta,\,u_\gamma,\dots$	Unsicherheiten der Fitparameter
$u_{\alpha,\beta},\dots$	Kovarianz zwischen α und β
$C_{\gamma,\alpha},\dots$	Korrelationskoeffizient zwischen α und γ

Python Code 9: Kovarianzmatrix & Korrelationsmatrix

Pythons **curve_fit** führt als zweites Rückgabelement die Kovarianzmatrix direkt auf. Die Korrelationsmatrix wird daraufhin durch manuelle Berechnung bestimmt:

```
import numpy as np
_, pcov = curve_fit(model, data_x, data_y)
errors = np.sqrt(np.diagonal(pcov))
corr = np.transpose(pcov / errors) / errors
print(pcov)
print(corr)
```

$$\begin{pmatrix} 1.24582399 \cdot 10^{-6} & 1.97394817 \cdot 10^{-6} & -2.57063649 \cdot 10^{-6} \\ 1.97394817 \cdot 10^{-6} & 3.99026378 \cdot 10^{-6} & -4.07304983 \cdot 10^{-6} \\ -2.57063649 \cdot 10^{-6} & -4.07304983 \cdot 10^{-6} & 6.74198784 \cdot 10^{-6} \end{pmatrix}$$

$$\begin{pmatrix} 1. & 0.88533289 & -0.88698922 \\ 0.88533289 & 1. & -0.78528072 \\ -0.88698922 & -0.78528072 & 1. \end{pmatrix}$$

8.6 Korrelationen linearer Funktionen

Führt man eine Regression über eine lineare Funktion der Form $f(x) = a \cdot x + b$ durch und benutzt beide Parameter a und b später um den Bruch b/a oder a/b zu berechnen, muss man das unter voller Beachtung der Korrelation tun. Es gibt aber einen Trick um sich die gesamte korrelierte Gaußsche Messunsicherheitsfortpflanzung zu sparen. Wenn man eine Variable aus

der Gleichung ausklammert:

$$f(x) = a \cdot \left(x + \frac{b}{a} \right) \tag{70}$$

$$f(x) = b \cdot \left(\frac{a}{b} \cdot x + 1 \right) \tag{71}$$

und den inneren Bruch-Term mit einem Parameter substituiert

$$f(x) = a \cdot (x + c) \tag{72}$$

$$f(x) = b \cdot (d \cdot x + 1) \tag{73}$$

erhält man mit $c = b/a$ oder $d = a/b$ die Brüche mit kompletter korrelierter Unsicherheitsabschätzung dessen Funktionen nur noch nichtlinear gefittet werden müssen.