Karlsruher Institut für Technologie

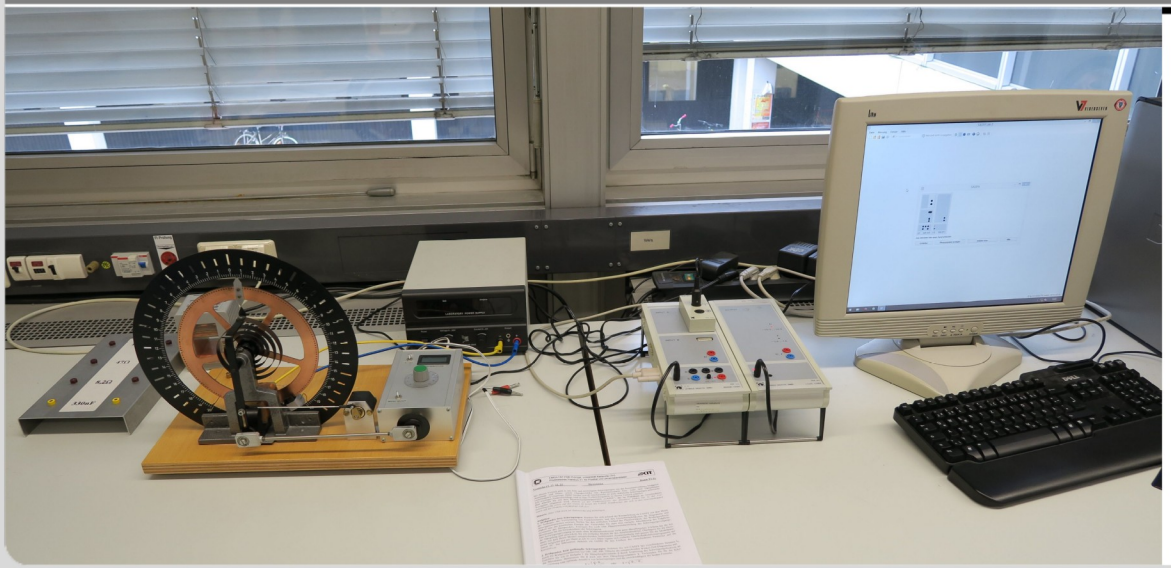
Institut für Experimentelle Teilchenphysik

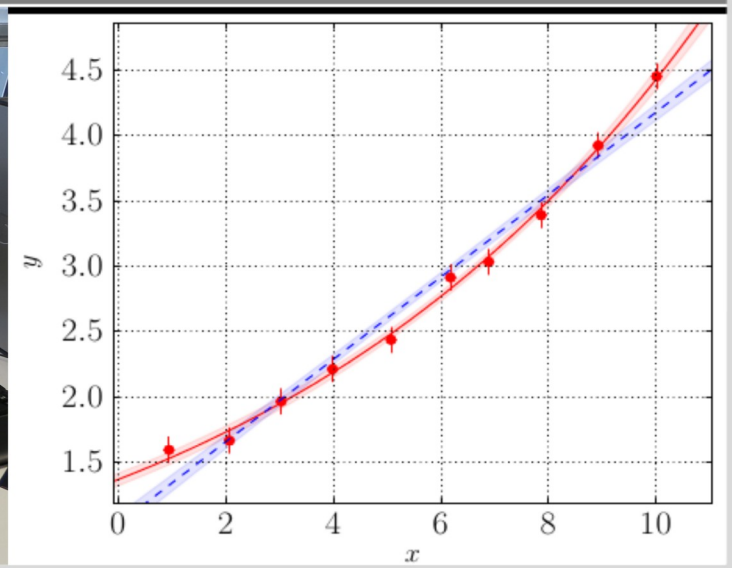
# Auswertung von Messdaten
## im Praktikum

**Prof. Dr. Günter Quast, Dr. Hans-Jürgen Simonis, PD Dr. Roger Wolf**

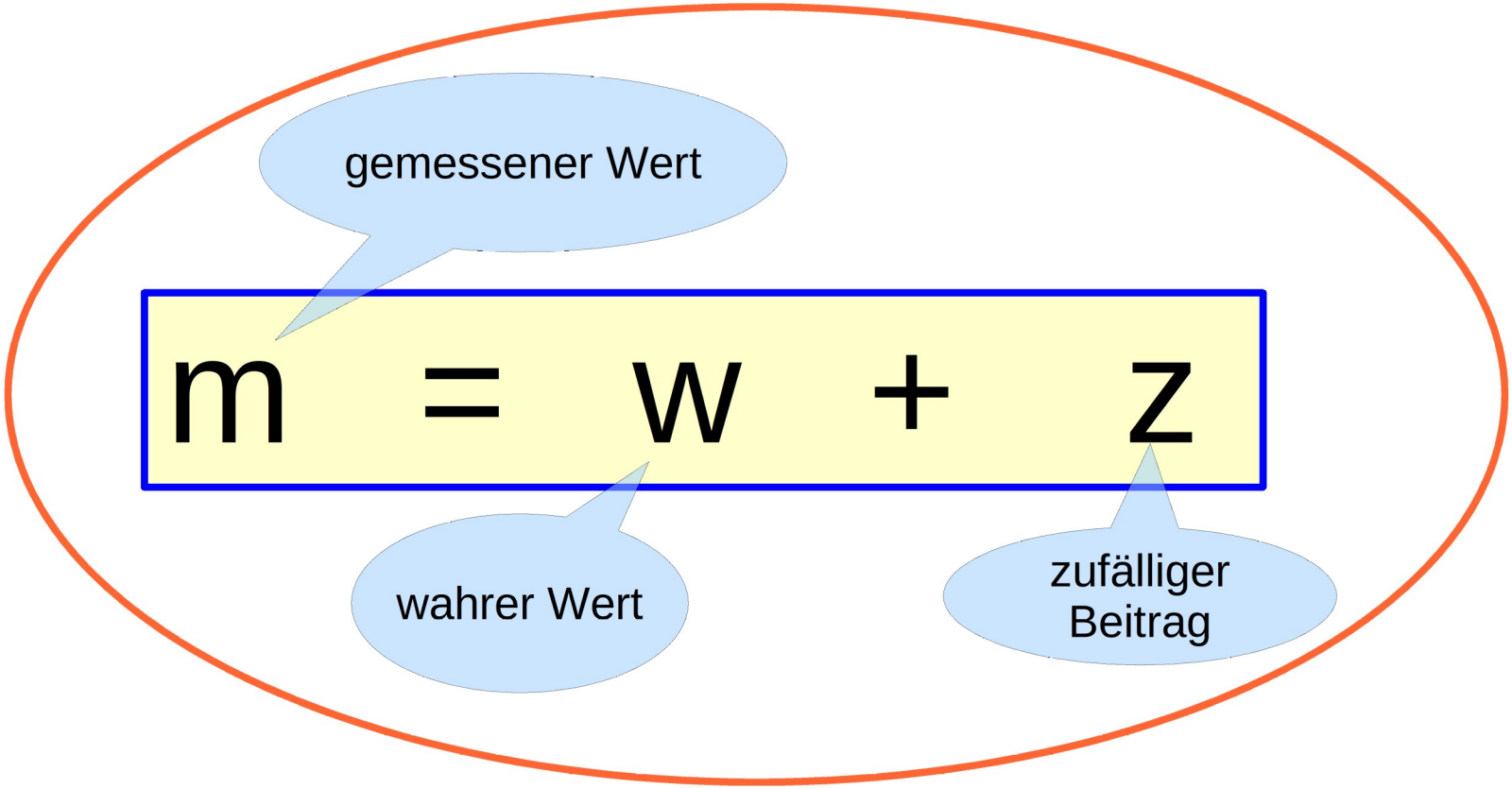
**WS 21/22**

Institut für Experimentelle
Teilchenphysik

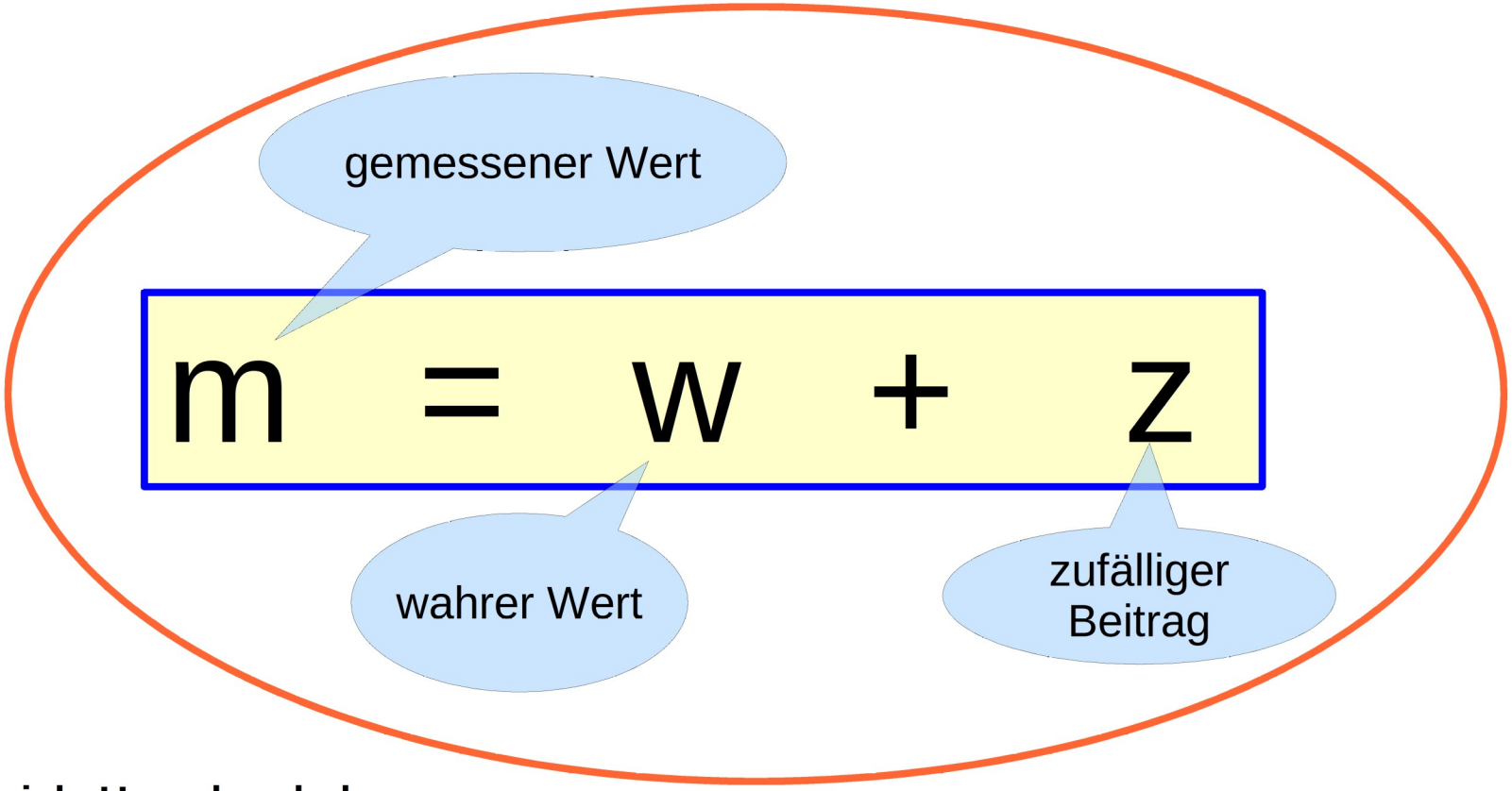




# Modell einer Messung



# Modell einer Messung




**z kann viele Ursachen haben:**

- Zufälliger Beitrag zum Messwert („Rauschen“) → „statistische Unsicherheit“
- Genauigkeit des verwendeten Messinstruments → „systematische Unsicherheit“
- Mitunter gibt es auch eine Unsicherheit auf den „wahren“ Wert, den man oft *z* zuschlägt → „theoretische Unsicherheit“

⇒ $z = z_{stat} + z_{syst} + z_{theo}$

Fehler im Messprozess – sollten nicht passieren !

# Darstellung von Messergebnissen

Messung einer physikalischen Größe x bedeutet, experimentell die **Maßzahl** zur Maßeinheit zu ermitteln:

$$G = (G) \cdot [G]$$

Dazu gehört auch die Angabe der **Messunsicherheit ΔG**:

$$G = (G \pm \Delta G) \cdot [G]$$

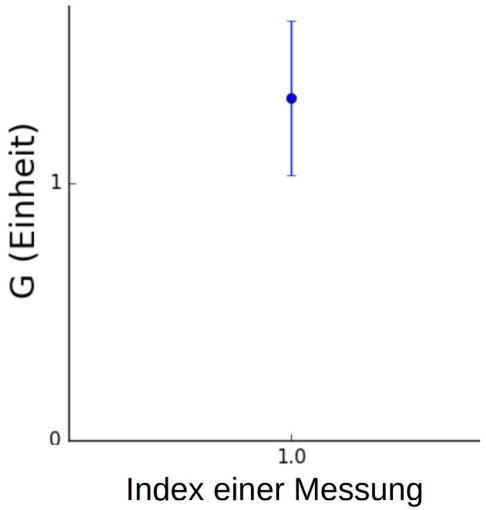
Üblich ist auch die Angabe einer **relativen Meßunsicherheit**:

$$G = (G) \cdot [G] \pm \Delta x / x$$

Δx/x wird meist in % angegeben, aber ggf. auch auch ‰ oder ppM („parts per Million“).

---

**Grafische Darstellung:**
Koordinatensystem mit Messpunkt und „Fehlerbalken“



# Beispiel einer Messung

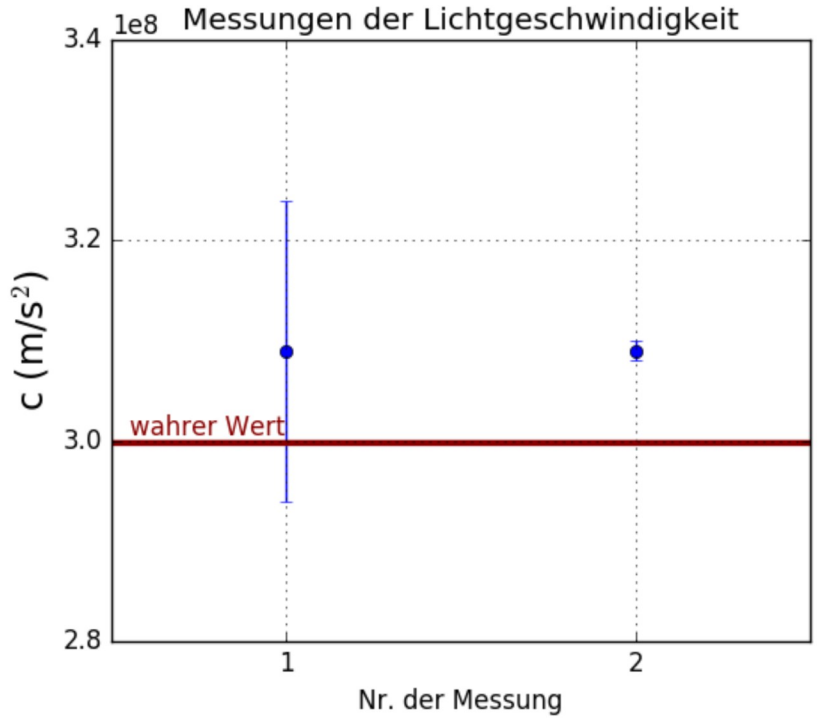
Entscheidend für die Bewertung eines Ergebnisses ist die **Messunsicherheit!**

**Beispiel:** Die Lichtgeschwindigkeit ist bereits sehr genau bekannt,
„Literaturwert“: $c = 2.99792458 \cdot 10^8$ m/s

Eine Messung $c = (3.09 \pm 0.15) \cdot 10^8$ m/s ist in **Übereinstimmung**

Eine Messung $c = (3.09 \pm 0.01) \cdot 10^8$ m/s wäre dagegen im **Widerspruch**

## Grafische Darstellung mit erzeugendem Skript in der Sprache python




```
import numpy as np, matplotlib.pyplot as plt
# die Messdaten
c_m=[3.09e8, 3.09e8]
c_e=[0.15e8, 0.01e8]
c_w=2.99792458e8
# grafische Darstellung
plt.errorbar([1, 2], c_m, yerr=c_e, fmt='bo')
plt.axhline(c_w, color='darkred', linewidth=3)
plt.text(0.55, 3.005e8, 'wahrer Wert',
color='darkred')
plt.ylabel("c (m/s$^2$)", size='x-large')
plt.xlabel("Nr. der Messung")
plt.title("Messungen der Lichtgeschwindigkeit")
# (... + einige Verschoenerungen ...)
plt.show()
```

**Angabe einer physikalischen Größe ohne Messunsicherheit ist wertlos!**

# Signifikante Stellen eines Ergebnisses

Nicht alle Stellen, die ein Rechner ausspuckt, sind relevant!

**Regel:**

Im Praktikum sollten die Messunsicherheiten („Fehler“) auf eine signifikante Stelle gerundet werden. Die letzte Stelle des Messwerts hat die gleiche Größenordnung.

**Ausnahme:**

Wenn ein Zwischenergebnis mit Unsicherheit weiter verwendet werden soll, so wird mindestens eine signifikante Stelle mehr mitgenommen

**Beispiel:**

Das numerische Ergebnis einer Messung der Erdbeschleunigung sei
$g = (9.8234 \pm 0.02385)\ m/s^2$

→ Angabe **$g = (9.82 \pm 0.02)\ m/s^2$**
bzw. **$g = 9.82\ m/s^2 \pm 0.2\ \%$**

**Hinweis:**

Übersichtliche Schreibweise ist wichtig!

$m = (0.0000082 \pm 0.0000003)$ kg ist zwar korrekt, aber schwer lesbar
$m = 0.0000082$ kg ± 0.3 mg ebenfalls

**$m = (8.2 \pm 0.3) \times 10^{-6}$ kg** oder **$m = (8.2 \pm 0.3)$ mg** ist **viel besser**!

# Fehleranalyse

Zu jeder Messung gehört eine **Fehleranalyse**: Ursache und Größe von Messabweichungen

**Ursachen:**

- Fehlerhafte Bedienung von Messgeräten (z.B. falsch kalibriert)?
- Irrtum beim Protokollieren oder der Auswertung (z.B. Zahlendreher)?
- Messverfahren oder Messbedingung ungeeignet?

Grobe Abweichungen sollten durch sorgfältiges Experimentieren und Kontrolle (möglichst durch eine zweite Person) vermieden werden!

Grob fehlerhafte Werte einer Messreihe sollten nicht weiter verwendet werden.

**Aber Achtung:**

*Eine nicht zu unterschätzende Fehlerquelle ist eine mangelnde Objektivität des Experimentators. Oft entstehen falsche Messresultate auch dadurch, dass der Experimentator das Resultat, das er haben will, aus unzureichenden Daten herausliest oder sogar Daten manipuliert.*

kleiner Exkurs: der Fall „Schön“ (DPA-Meldung vom 11.06.2004)
Die **Universität Konstanz entzieht dem Physiker Jan Hendrik Schön seinen Doktortitel!**

Die Universität bezieht sich ausdrücklich nicht auf Fehler in seiner Doktorarbeit, sondern stuft Datenmanipulationen während seiner späteren Forschertätigkeit an den Bell-Labs in den USA als wissenschaftlich unwürdiges Handeln ein. Das baden-württembergische Universitätsgesetz lässt einen Titelentzug auch auf Grund späteren unwürdigen Verhaltens zu.

# Systematische Unsicherheiten

**„Systematische Fehler“** zeigen bei identischen Messbedingungen immer um den gleichen Betrag in die gleiche Richtung.

*Können durch Messwiederholung weder erkannt noch eliminiert werden; beeinflus-sen alle unter gleichen Bedingungen erfolgten Messungen in gleicher Weise; sind erfassbar durch Variation der Messmethode oder der Messbedingungen.*

| Ursachen | Beispiele |
|---|---|
| Fehlerhafte Meßgeräte | Eich- oder Justierfehler, Drift,... |
| Umwelteinflüsse | Temperatur, Druck,... |
| Rückwirkung der Meßgeräte | Innenwiderstand, Verformung, .. |
| Unzulänglichkeit des Experimentators | |
| Gültigkeitsgrenzen der phys. Gesetze | |

Erkannte systematische Probleme können und müssen korrigiert werden!

**Beispiel:** Temperaturausdehnung eines Maßstabes, geeicht bei 20°, verwendet bei 30°:

- Relative Temperaturausdehnung $\alpha = 0.0005/K$
- Korrektur $L' = L \cdot [1 + 0.0005 \cdot (30-20)] = 1.005 \cdot L$

Aus mess- oder rechentechnischen Gründen nicht erfassbare systema-tische Unsicherheiten müssen abgeschätzt werden.

# Statistische (=zufällige) Unsicherheiten

**„Statistische Fehler“** beeinflussen Messergebnisse trotz identischer Bedingungen unterschiedlich in Betrag und Vorzeichen.

*Sind zufällig in dem Sinne, dass ihre Ursachen im Einzelnen nicht verfolgt werden können („Zufall“ durch Unkenntnis in der klassischen Physik) bzw. als Eigenschaft des Messprozesses in der Quantenphysik; sind unvermeidbar und werden mit Methoden der Statistik behandelt.*

| Subjektive Ursachen | Objektive Ursachen |
|---|---|
| Parallaxenfehler | Äußere Einflüsse (p, T) |
| Skaleninterpolation | Statistische Messgröße (Rauschen) |
| Reaktionsvermögen | |

**Zufällige Messunsicherheiten** lassen sich durch **Messwiederholung** und **Mittelung** reduzieren.

Bei $n$ Wiederholungen einer Messung mit Einzelunsicherheit $\Delta x$

gilt für die Unsicherheit des Mittelwerts $\Delta\bar{x} = \frac{\Delta x}{\sqrt{n}}$

# Zentrale Frage…

Wie bestimmt man die Werte der Messunsicherheiten?

Wie sind sie zu interpretieren?

- einmalige Messungen
  → Fehler abschätzen
- wiederholte gleichartige Messungen
  → statistische Auswertung
- aus Messgrößen berechnete Ergebnisse
  → Fehlerfortpflanzung

# Beispiel: einmalig gemessene Größen

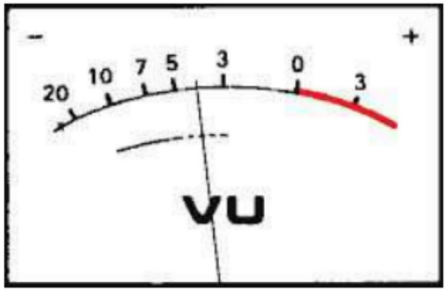
Bei einmalig gemessenen Größen schätzt man die Unsicherheit.

Grobe Richtlinien:

a) bei fein unterteilter Skala → ± 0.5 · (Intervallbreite)

L = (82 ± 0.5) mm

b) bei grob unterteilter Skala → ± 0.1 · (Intervallbreite)




U = (4.0 ± 0.2) V

c) digitale Skala → ± 0.5 · letzte Anzeigestelle
⊕ Geräteklasse (lt. Datenblatt)
⊕ evtl. Anzeigefluktuationen

# Beispiel: Messreihen

Bei Messreihen (mit N Werten $x_i$) führt man eine **statistische Analyse** durch.

**Sprechweise der Statistik:**
Eine Messreihe stellt eine Stichprobe aus der Menge der möglichen Messwerte (Grundgesamtheit) dar. Die Häufigkeitsverteilung der Einzelmessungen nähert sich mit steigender Stichprobenlänge der Verteilungsdichte der Grundgesamtheit an.

Unter schwachen Voraussetzungen (→ zentraler Grenzwertsatz) ist die zugrunde liegende Verteilungsdichte in der Praxis häufig die **Normal-** oder **Gauß-Verteilung:**

Erwartungswert (μ):

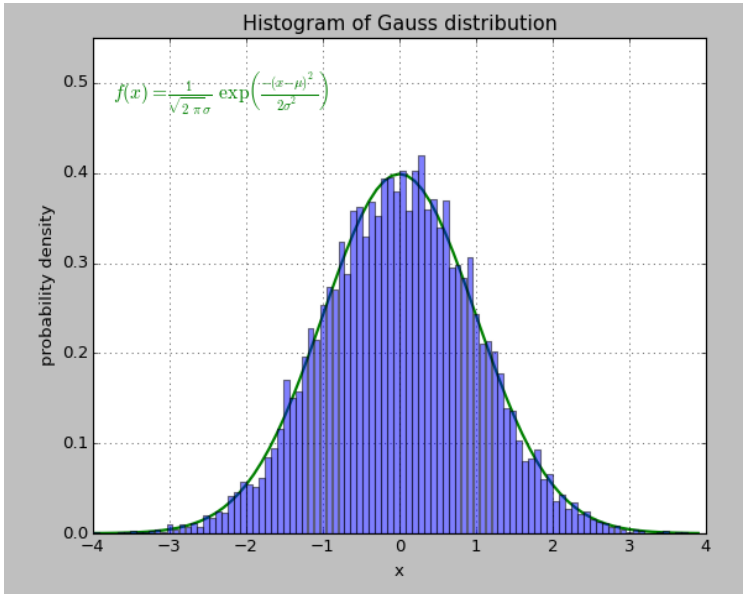
$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

Standardabweichung (bei unbekanntem μ):

$$\sigma_x = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2}$$

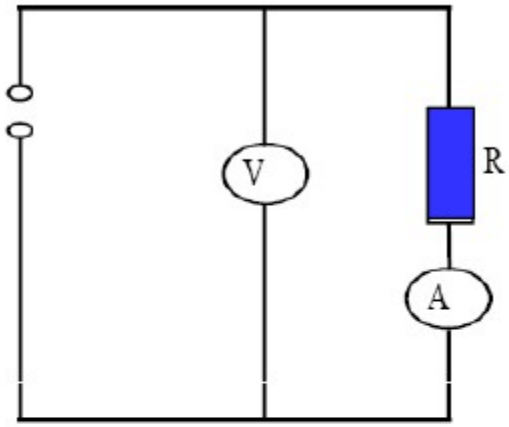
Standardabweichung des Mittelwerts:

$$\sigma_{\bar{x}} = \frac{\sigma_x}{\sqrt{N}}$$




Skript animated_Gauss.py

# Beispiel: Messung eines Widerstands R




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

| U \ V | I \ mA | R=U/I (Ω) |
|---|---|---|
| 7,8 | 35 | 222.86 |
| 15,6 | 65 | 240.00 |
| 23,4 | 78 | 300.00 |
| 31,3 | 126 | 248.41 |
| 39,0 | 142 | 274.65 |
| 46,9 | 171 | 274.27 |
| 54,7 | 194 | 281.96 |
| 62,6 | 226 | 276.99 |
| 78,3 | 245 | 319.59 |
| 86,0 | 258 | 333.33 |
| 87,6 | 258 | 339.53 |
| 93,6 | 271 | 345.39 |
| 101,6 | 277 | 366.79 |
| 109,6 | 284 | 385.91 |
| 118,0 | 290 | 406.90 |

N = 15 Messungen

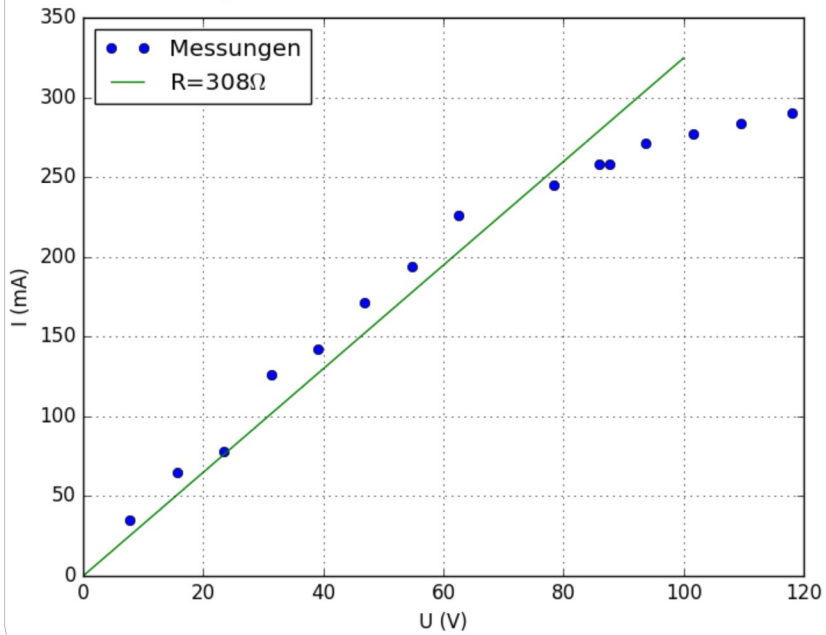
# Beispiel: Messung eines Widerstands R

Nach Kochrezept: $R = \frac{1}{N}\sum R_i = 307.77\,\Omega\,,\ \Delta R = \sqrt{\frac{\sum (R_i - R)^2}{N(N-1)}} = 14\,\Omega$

$$\Rightarrow\quad R = (308 \pm 14)\,\Omega$$

Bei genauerem Hinsehen:

Grafische Darstellung von Daten und Ergebnis




```
import numpy as np, matplotlib.pyplot as plt

U=[7.8,15.6,23.4,31.3,39.0,46.9,54.7,62.6,78.3,86.0,
    87.6,93.6,101.6,109.6,118.0]
I=[35,65,78,126,142,171,194,226,245,258,258,271,
    277,284,290]

plt.plot(U, I, 'bo',  label = "Messungen")
plt.xlabel("U (V)")
plt.ylabel("I (mA)")
x=np.arange(0, 140., 100)
plt.plot(x, 1/0.308*x,'g-',label = "R=308$\Omega$")
plt.legend(loc='best')
plt.grid()
plt.show()
```

Skript zur Erzeugung der Grafik

Hier ist etwas faul … ?!

**Anm:** Mittelung nicht verträglicher Werte ist unsinnig! (Hier kam es zur Erwärmung des Widerstands. Dies führt zu Abweichungen vom naiven Ohm'schen Gesetz)

# Häufigkeiten und Histogramm-Darstellung

Auswertung umfangreicher Messreihen:

Einteilung der *N* Messungen in *k* „Klassen“ (z.B. Intervalle 1, …, *k*) und Häufigkeit des Vorkommens auftragen.

**Histogrammdarstellung:** „Balkendiagramm“ der Häufigkeiten $H_i$, *i =1, …, k*

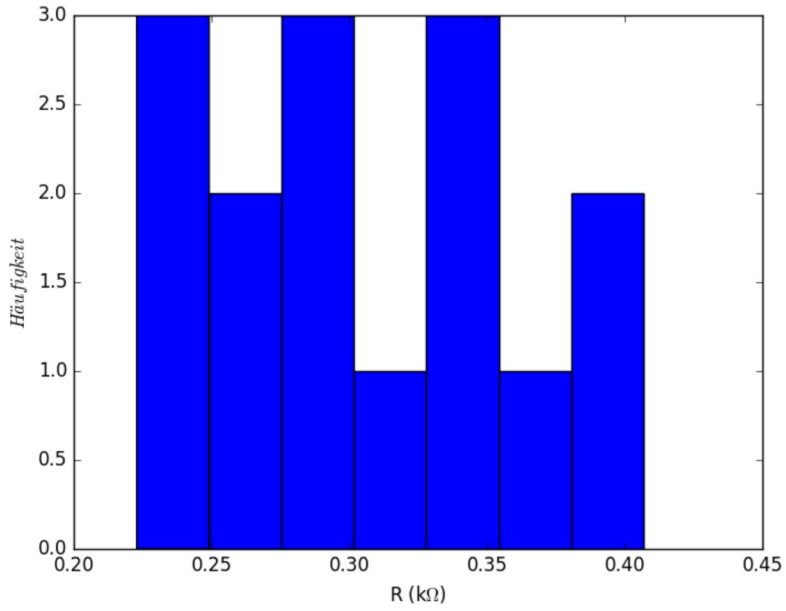
Häufigkeiten: $\sum_{i=1}^{k} H_i = N$

relative Häufigkeiten: $h_i = \frac{H_i}{N} \Leftrightarrow \sum h_i = 1$

Mittelwert der Messgrößen: $\overline{x} = \frac{1}{N} \sum x_i$

Messunsicherheit: $\sigma_x = \frac{1}{N-1} \sum (x_i - \bar{x})^2$

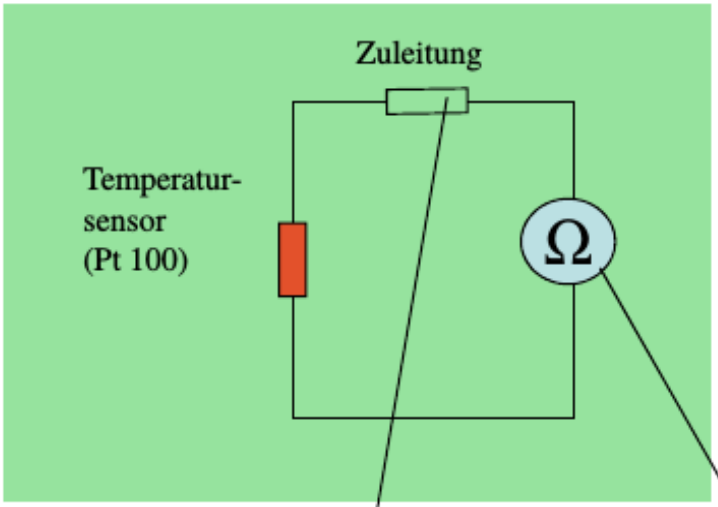
**Beispiel**: Widerstandsmessungen von eben als Häufigkeitsdiagramm (→ Histogramm)




```
import numpy as np, matplotlib.pyplot as plt

U=np.array([7.8,15.6,23.4,31.3,39.0,46.9,54.7,6
2.6,

78.3,86.0,87.6,93.6,101.6,109.6,118.0])
I=np.array([35,65,78,126,142,171,194,226,245,2
58,
                258,271,277,284,290])
R=U/I
plt.hist(R,7)
plt.xlabel("R ($\Omega$)")
plt.ylabel(r"$H\"aufigkeit$")
plt.show()
```

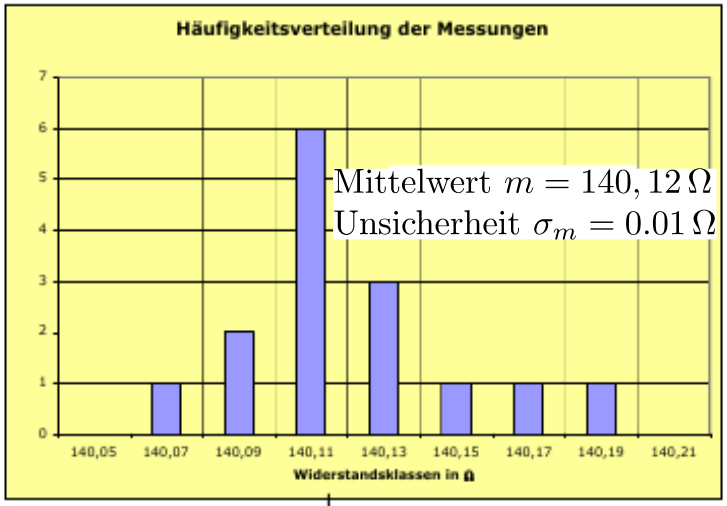
Skript Histogram.py

# Beispiel: Wiederholte Widerstandsmessung




| Klasse/Ω | Häufigkeit |
|---|---|
| 140.05 | 0 |
| 140.07 | 1 |
| 140.09 | 2 |
| 140.11 | 6 |
| 140.13 | 3 |
| 140.15 | 1 |
| 140.17 | 1 |
| 140.19 | 1 |
| 140.21 | 0 |

Summe = 15




**Systematische Abweichung: Zuleitungswiderstand 1,00 Ω**

**Herstellergarantie 0,05 Ω**

**statistische Unsicherheit 0,01 Ω**

**Messergebnis: 140,12 Ω**

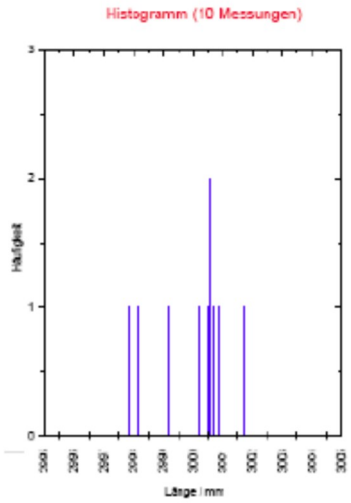
Korrektur: Messergebnis - Abweichung
139,12 Ω

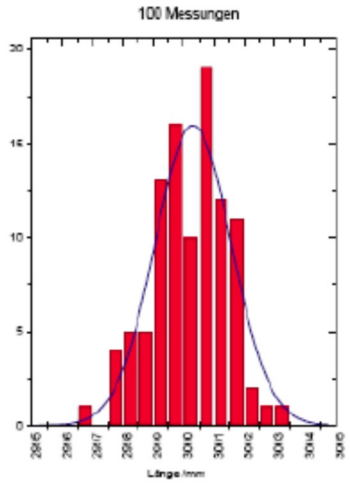
**Endergebnis: (139,12 ± 0,01 ± 0,05) Ω**

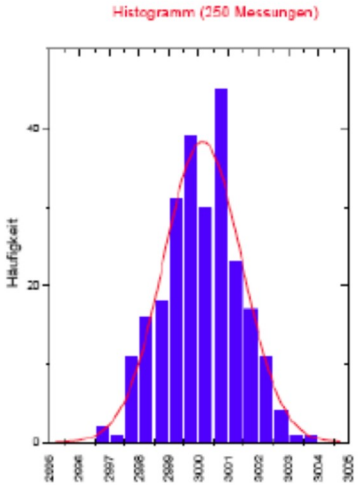
**Falls die Gesamtunsicherheit gefordert wird: quadratische Addition**

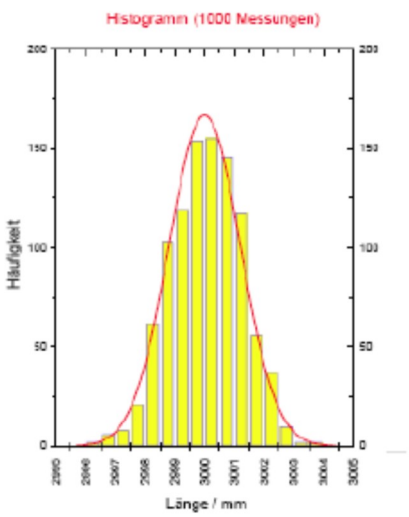
# Grenzverteilungen

Mit zunehmender Anzahl von Messungen nähert sich die **Häufigkeitsverteilung** einer **kontinuierlichen Grenzverteilung** an.



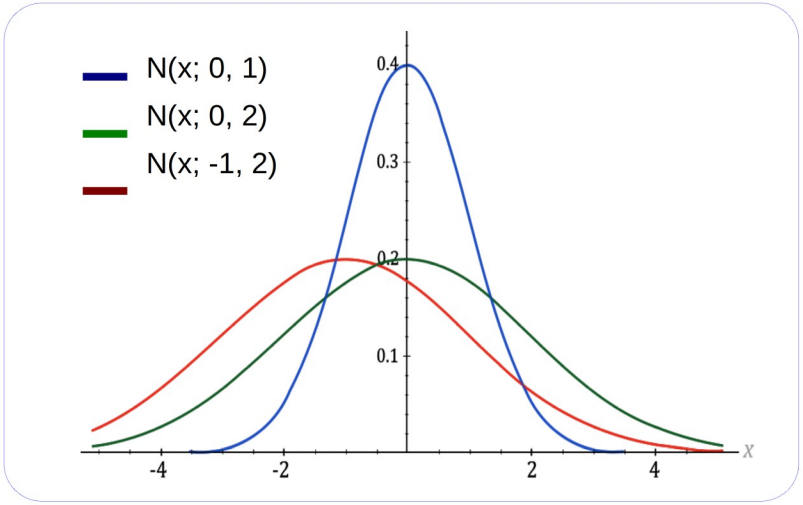




Unter schwachen Annahmen (Lyapunov-Bedingung) ist die Grenzverteilung der statistischen Abweichungen die Gauß-Verteilung:

$$N(x;\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} \mathrm{e}^{\frac{(x-\mu)^2}{2\sigma^2}}$$

(zentraler Grenzwertsatz der Statistik)



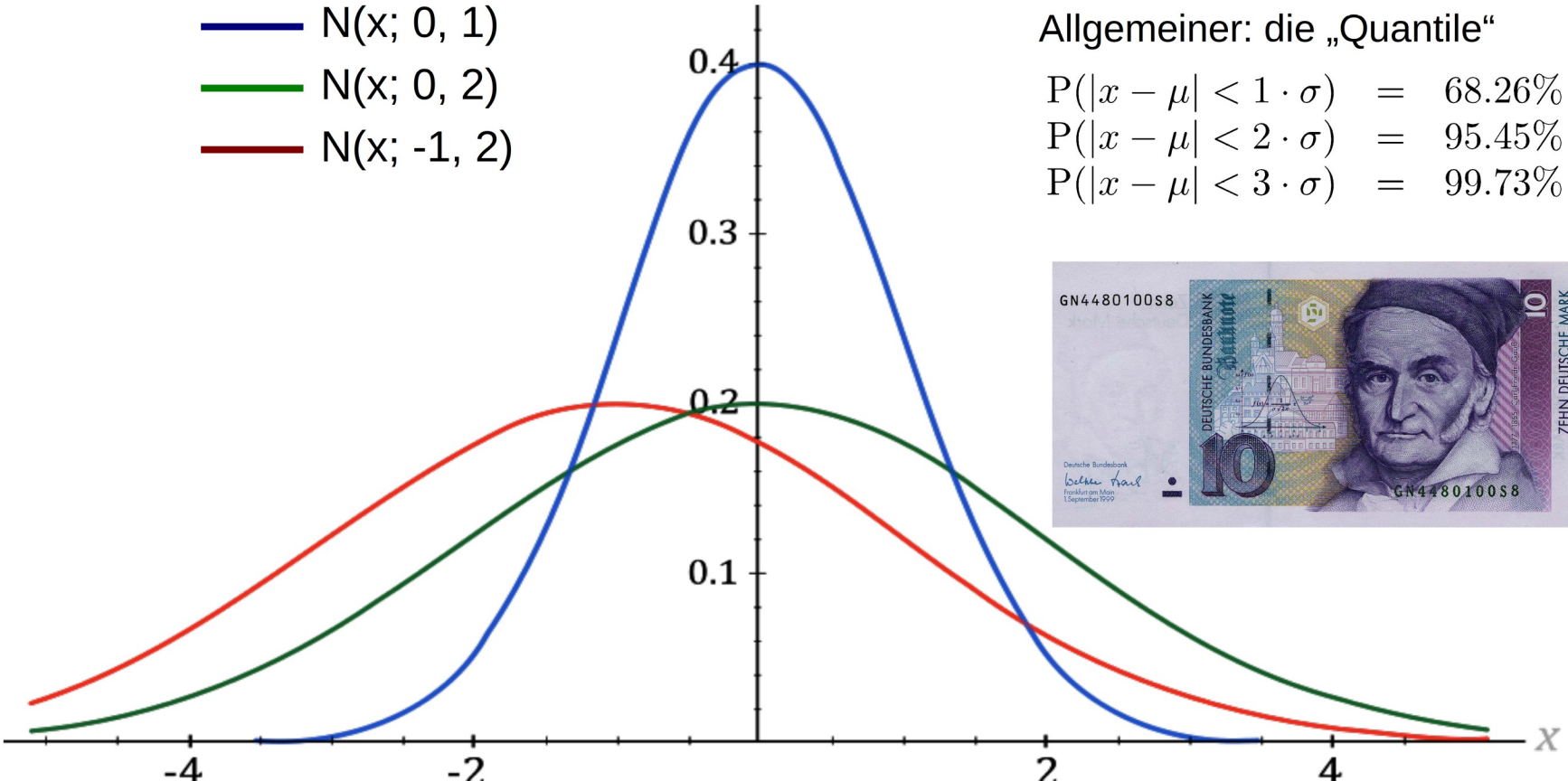
# Eigenschaften der Gauß- oder Normalverteilung

$$N(x;\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} \mathrm{e}^{\frac{(x-\mu)^2}{2\sigma^2}}$$

Erwartungswert $\mathrm{E}[x] = \mu$

Standardabweichung $\sigma$
(Maß für die Breite, entspricht der Messunsicherheit Δx von zuvor)

68,3% aller Messungen liegen im Intervall $[\mu - \sigma,\ \mu + \sigma]$




Allgemeiner: die „Quantile“

$$\begin{aligned} \mathrm{P}(|x-\mu| < 1\cdot\sigma) &= 68.26\% \\ \mathrm{P}(|x-\mu| < 2\cdot\sigma) &= 95.45\% \\ \mathrm{P}(|x-\mu| < 3\cdot\sigma) &= 99.73\% \end{aligned}$$

# Gewichteter Mittelwert

Wenn eine Größe mehrfach (Werte $x_i$) mit unterschiedlichen Unsicherheiten ($\sigma_i$) bestimmt wurde, so bildet man einen „**gewichteten Mittelwert**“.

Die Gewichte sind dabei die quadrierten Kehrwerte der Unsicherheiten:

$$\bar{x} = \frac{\sum w_i \, x_i}{\sum w_i} \quad \text{mit} \quad w_i = 1/{\sigma_i}^2$$

Präzisere Messungen erhalten das größere Gewicht.

**Anm.:** Überlegen Sie, was die quadratische Wichtung für Ihre Arbeit bedeutet, wenn Sie eine Größe mit dem dreifachen der bisher bekannten Unsicherheit bestimmen.

Die **Unsicherheit** des gewichteten Mittelwerts ergibt sich folgendermaßen:

$$\sigma_{\bar{x}} = 1 \Big/ \sqrt{\sum w_i} \quad \left( \Leftrightarrow \quad \frac{1}{\sigma_{\bar{x}}^2} = \sum \frac{1}{{\sigma_i}^2} \right)$$

(Beweis z.B. mit Hilfe der Fehlerfortpflanzung, s.u., bzw. Vorlesung CgDA)

python Skript:

```
import numpy as np

w = 1/sx**2
sumw = np.sum(w)
mean = np.sum(w*x)/sumw
smean = np.sqrt(1./sumw)
```

# Fehlerfortpflanzung

Wenn eine Größe y nicht direkt messbar ist, sondern als Funktion von anderen (Mess-)Größen abhängt, also $y = f(x_1, \ldots, x_n)$ , dann wendet man das **Gauß'sche Fehlerfortpflanzungsgesetz** an:

$$\sigma_y^2 = \sum_{i=1}^{n} \left( \frac{\partial y}{\partial x_i} \right)^2 \sigma_{x_i}^2$$

Dabei wird die **statistische Unabhängigkeit** der Größen $x_i$ vorausgesetzt. Für Nicht-lineare Funktionen ***f*** muss die Gültigkeit der **Taylor-Näherung** um die Bestwerte der $x_i$ innerhalb der Unsicherheiten $\sigma_{xi}$ gewährleistet sein. Wenn diese Voraussetzungen nicht gegeben sind, müssen andere Methoden angewandt werden (Berücksichtigung der Kovarianz-Matrix oder Monte-Carlo-Methode, s. Vorl. CgDA).

Spezialfälle:

$y = x_1 + x_2$ oder $y = x_1 - x_2$

$\Rightarrow \sigma_y{}^2 = \sigma_1{}^2 + \sigma_2{}^2$

Quadrierter absoluter Fehler auf die Summe oder Differenz zweier Messungen ist die quadratische Summe ihrer absoluten Fehler

$y = x_1 \cdot x_2$ oder $y = \frac{x_1}{x_2}$

$\Rightarrow \left(\frac{\sigma_y}{y}\right)^2 \simeq \left(\frac{\sigma_1}{x_1}\right)^2 + \left(\frac{\sigma_2}{x_2}\right)^2$

Quadrierter relativer Fehler auf das Produkt oder Verhältnis zweier Messungen ist die quadratische Summe ihrer relativen Fehler

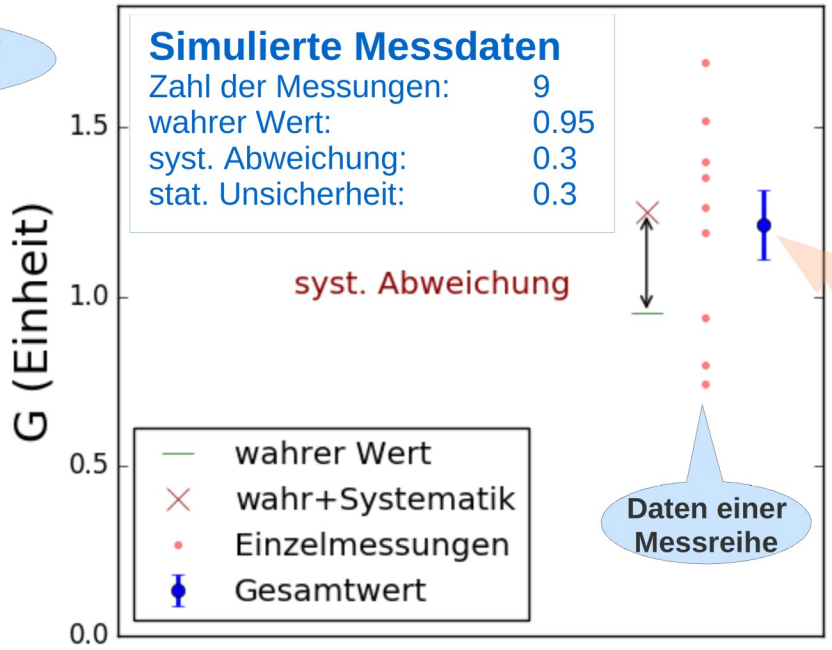
# Zusammenfassung: Messung

Messwert — **wahrer Wert** — **zufällige** statistische Abweichung

$$m = w + z_{sys} + z_{stat}$$

**zufällige** systematische Abweichung

- systematische Unsicherheit betrifft alle Messwerte in gleicher Weise
- statistische Unsicherheit ist bei jeder Messung anders




Statistische Unsicherheiten können durch Mehrfachmessung und Mittelwertbildung reduziert werden → Zusammengefasst als

Messpunkt mit „Fehlerbalken“

$$\sigma_{\bar{G}} = \frac{\sigma_G}{\sqrt{N}}$$

Gesamtunsicherheit: $\sigma_{\mathrm{tot}} = \sqrt{\sigma_{\mathrm{stat}}^2 + \sigma_{\mathrm{sys}}^2} = \sqrt{0.3^2/9 + 0.3^2} = 0.32$

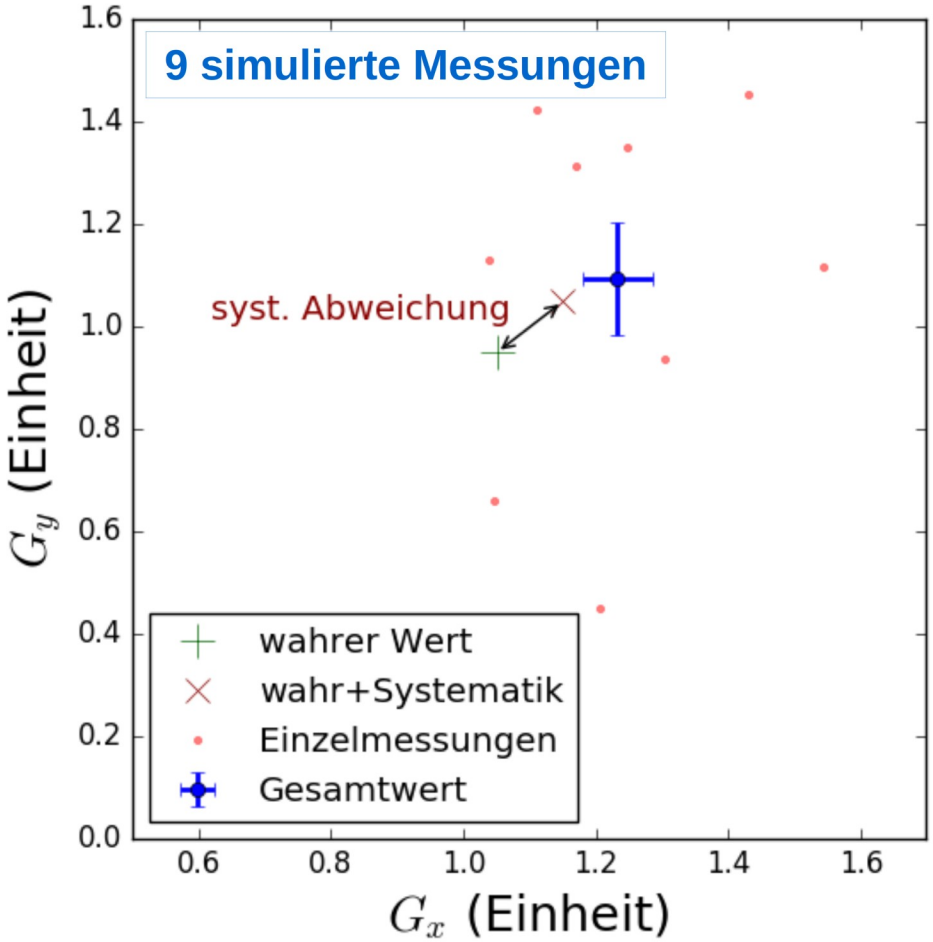
dominiert durch systematische Unsicherheit!

- da verschiedene Unsicherheiten laut **Fehlerfortpflanzungsgesetz** quadratisch addiert werden, können kleine Beiträge im Endergebnis vernachlässigt werden
  → Konzentratieren Sie sich auf die dominanten Effekte!

# Paare von Messungen

Sehr häufig werden Paare von Messungen $(G_x, G_y)$ aufgenommen. Wenn diese Messungen unabhängig sind lassen sich die Überlegungen verallgemeinern: aus dem „Fehlerbalken“ wird ein Fehlerkreuz.

Mittelwert, Unsicherheit auf den Mittelwert und Fehler-fortpflanzung in jeder Koor-dinate separat berechnen.




**Anm.:** das Fehlerkreuz zeigt nur die statistischen Unsicherheiten

# Datenquellen

**Daten aus Messungen** können in unterschiedlicher Form gewonnen werden:

- Ablesen von analogen und digitalen Messgeräten
  Eingabe über Tastatur zur Darstellung / Auswertung
- Datenexport aus digitalen Messgeräten, insb. „Datenlogger“ oder Digitaloszilloskope
  Typischerweise große Datenmengen erfordern programmgestütztes Einlesen sowie Funktionen zur Darstellung und Signalverarbeitung (Maxima/Minima, Periodendauer bzw. Frequenz, Frequenzspektrum, ...)
- Weiterverarbeitung der Ausgabe von Analyseprogrammen
  Erfordert Funktionen zur Datenübergabe an Programme zur finalen Auswertung und Darstellung der Ergebnisse

# Übliche Datenformate

Messgeräte (auch „Datenlogger“) und einige Handy-Apps (z.B. phyphox) nutzen einfache Datenformate in Text-Form:

Beispiel PicoScope, „Comma Separated Values“ (CSV)

```
Time,Channel A
(ms),(V)
```
} Kopfzeilen mit sog. „**Meta-Daten**“

```
-0.349,-0.000458
-0.348,-0.000458
-0.347,-0.000458
    . . .
```
} die eigentlichen **Daten** als Dezimalzahlen, in Spalten durch „,“ getrennt

Üblich sind auch „Tabulator-getrennte“ Dezimalzahlen und bisweilen auch Dezimalzahlen mit „**,**“ statt „**.**“ (dann müssen für die Verwendung in python-Programmen Dezimalkommata durch Dezimalpunkte ersetzt werden!)

Durchgesetzt haben sich „beschreibende“ Formate, z.B.
**xml** = „**e**xtensible **m**arkup **l**anguage“ oder
**json** = „**j**ava **s**cript **o**bject **n**otation“ (python dict);
gut durch python-Module unterstützt !

Rein „binäre“ Datenformate (also sehr kompakte Darstellungen in maschinenabhängigem digitalem Format) werden wegen ihrer Plattformabhängigkeit heute kaum noch verwendet.

Beispiel-Code

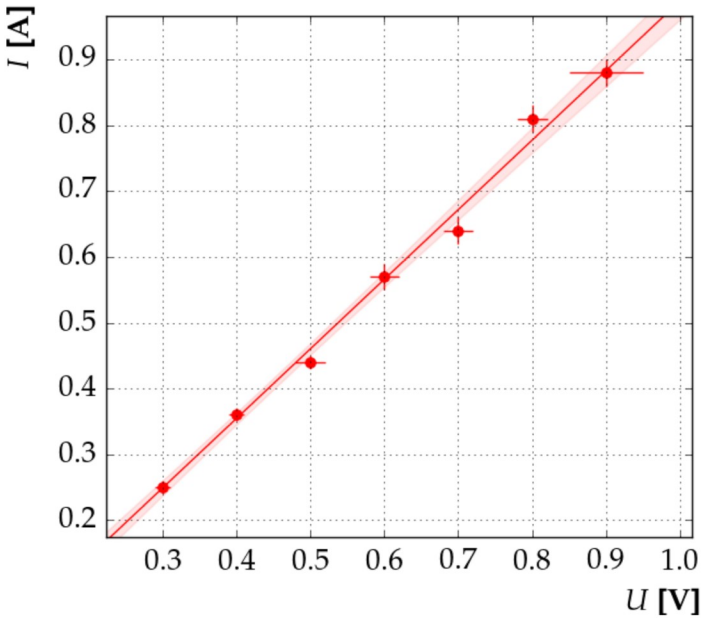
```
    Daten im csv-Format lesen

    Datei zum Lesen öffnen
f = open('AudioData.csv', 'r')

    Kopzeile(n) lesen
header=f.readline()
print "Kopfzeile:", header

    Daten in    D-numpy-array einlesen
data = np.loadtxt(f,
      delimiter=',', unpack=True)
print "-> Anzahl Spalten",
      data.shape[0]
print "-> Datenzeilen",
      data.shape[1]

    Daten in    D-arrays speichern
t = data[0]
a = data[1]
l = len(a)
```

# Modellanpassung/Regression

In der Regel gibt es einen funktionalen Zusammenhang zwischen (Mess-)Größen:

$y = f(x;\ p_1, \ldots,\ p_n)$ .

Die gesuchten physikalischen Größen stecken dann in den Parametern $p_1, \ldots, p_n$.




Beispiel von eben $(I, U)$ : $I = (1 / R) \cdot U$

**In aller Kürze:**

Mit numerischen oder analytischen Methoden werden die Parameter $p_k$ so bestimmt, dass ein vorgegebenes „**Abstandsmaß**“ zwischen den Messpunkten $y_i$ und den Funktionswerten $f_i = f(x_i; p_1, \ldots, p_n)$ **minimal** wird.

Script: Anpassen von Funktionen an Messdaten

15. November 2013

**Funktionsanpassung mit der $\chi^2$-Methode**

http://www.ekp.kit.edu/~quast

Vorschlag von Gauß:

Summe der kleinsten Fehlerquadrate,

$$S = \chi^2 = \sum_{i=1}^{N} \frac{(y_i - f(x_i, \{p\}))^2}{{\sigma_i}^2}$$

wird bzgl. der Parameter {p} minimiert.

# Minimieren der Größe S

**Minimierung von** $S = \chi^2 = \sum_{i=1}^{N} \frac{(y_i - f(x_i, \{p\}))^2}{{\sigma_i}^2}$ (N Messungen k Parameter)

- analytisch: $\frac{\partial S}{\partial p_j} = 0\,,\ j = 1, \ldots, k$ (Notwendige Bedingung für ein Minimum)

  lösbar in Spezialfällen, z. B. für lineare Probleme $f(x, \{p\}) = \sum_{j=0}^{k} p_j\, f_j(x)$

- i.a. numerische Optimierung: Algorithmen zur Suche nach dem(/einem) Minimum einer skalaren Funktion im *k*-dimensionalen Parameterraum

In der Praxis werden heute selbst für lineare Probleme numerische Minimierungsmethoden verwendet (außer in Spezialfällen, z.B. bei zeitkritischen oder immer wieder vorkommenden Problemstellungen).

# Beispiel: Minimierung mit einem Parameter

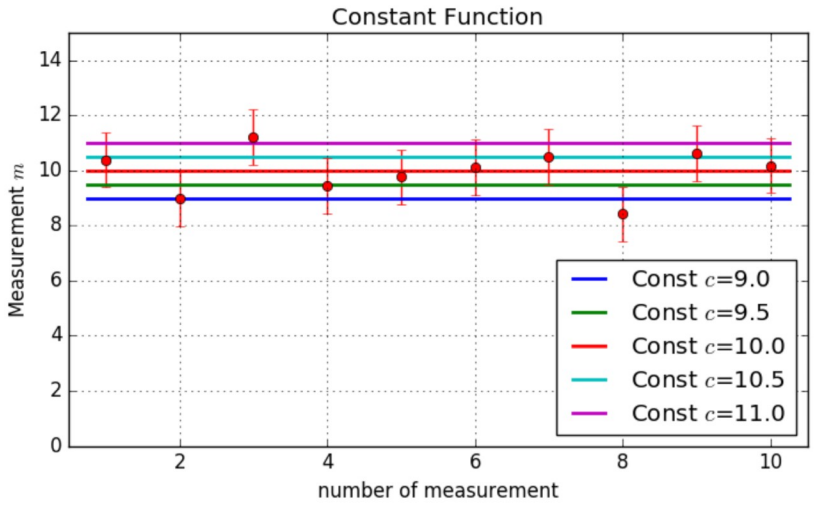
**Der Mittelwert** von 10 Messungen $y_i$ mit Unsicherheiten $\sigma$ entspricht der Anpassung einer konstanten Funktion *f(x;c)=c*

$$S(c) = \sum_{i=1}^{10} \frac{(y_i - c)^2}{\sigma^2}$$

analytisch: $0 = \frac{dS}{dc} = \sum_{i=1}^{N=10} \frac{-2(y_i - c)}{\sigma^2}$

$$\Rightarrow \hat{c} = \frac{1}{N} \sum_{i=1}^{N=10} y_i$$

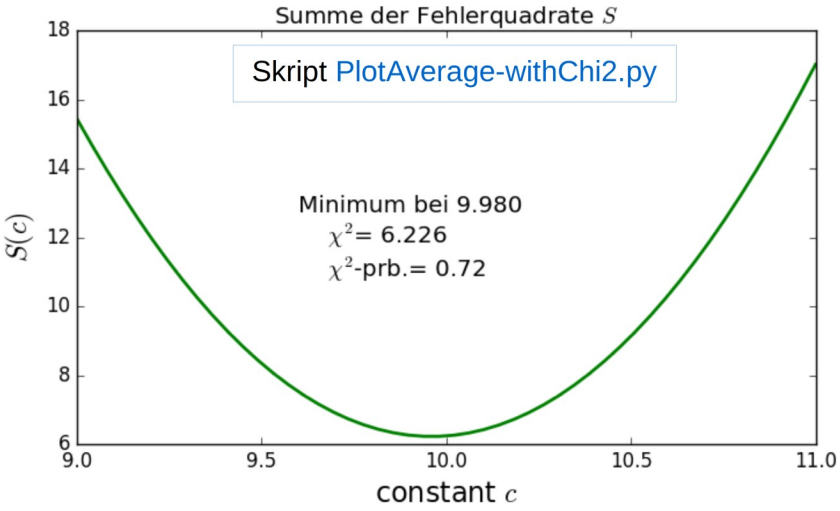
identisch zum „Mittelwert“




numerisch:

$$S(c) = \sum_{i=1}^{10} \frac{(y_i - c)^2}{\sigma^2}$$

berechnen und grafisch darstellen



# Beispiel: Geradenanpassung

$$f(x; p_1, p_2) = p_1 + p_2\, x \quad \Rightarrow \quad S = \sum_{i=1}^{N} \frac{(y_i - p_1 - p_2\, x_i)^2}{{\sigma_i}^2}$$

Nullsetzen der 1. Ableitungen:

$$(1)\ 0 \stackrel{!}{=} \frac{\partial S}{\partial p_1} = -2 \sum_{i=1}^{N} \frac{(y_i - p_1 - p_2\, x_i)}{{\sigma_i}^2}$$

$$(2)\ 0 \stackrel{!}{=} \frac{\partial S}{\partial p_2} = -2 \sum_{i=1}^{N} \frac{x_i\,(y_i - p_1 - p_2\, x_i)}{{\sigma_i}^2}$$

mit den Abkürzungen

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

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

ergibt sich die Lösung:

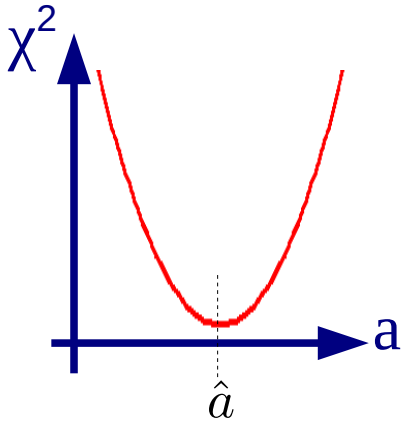
$$\hat{p}_1 = \frac{S_{xx} S_y - S_x S_{xy}}{D}\,, \qquad {\sigma_{p_1}}^2 = \frac{S_{xx}}{D}\,,$$

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

Implementiergung in python s. Script `linRegression`

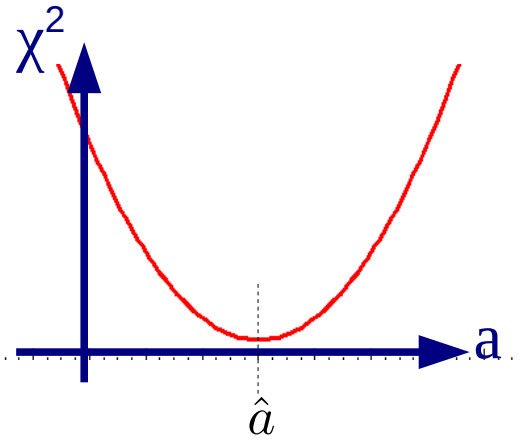
Diese Formeln waren für Generationen von Studierenden die Basis einer jeden Anpassung („Regression“), **aber** schon die Behandlung von Unsicherheiten in Ordinate **und Abszisse** erfordert numerische Methoden

# Numerische Bestimmung der Unsicherheiten




scharfes Minimum
→ große Krümmung

Je schärfer das Minimum von $\chi^2(\boldsymbol{p})$, desto kleiner die Parameterfehler:




flaches Minimum
→ kleine Krümmung

→ Parameterunsicherheiten sind umgekehrt proportional zu Krümmung(en) von $\chi^2(\boldsymbol{p})$ am Minimum

$$\frac{1}{\sigma_{\hat{p}}{}^2} = \frac{1}{2}\left.\frac{\partial^2\chi^2(p)}{\partial p^2}\right|_{\hat{p}}$$

bzw.

$$\left(V_{\hat{a}}{}^{-1}\right)_{ij} = \frac{1}{2}\left.\frac{\partial^2\chi^2(\{\mathbf{p}\})}{\partial p_i \partial p_j}\right|_{\hat{p_i}\hat{p_j}}$$

bei mehreren Parametern.

V: Kovarianzmatrix der Parameterunsicherheiten, s. Vorlesung CgDA

# Numerische vs. analystische Parameterbestimmung

**Anmerkung:**

Die Berücksichtigung von Unsicherheiten in Ordinaten- und Abszissenrichtung oder die Anpassung von Modellen, die nicht linear von den Parametern abhängen, sind analytisch nicht möglich.

Die **Empfehlung** ist daher, grundsätzlich Anpassungen mit **numerischen Werkzeugen** durchzuführen.

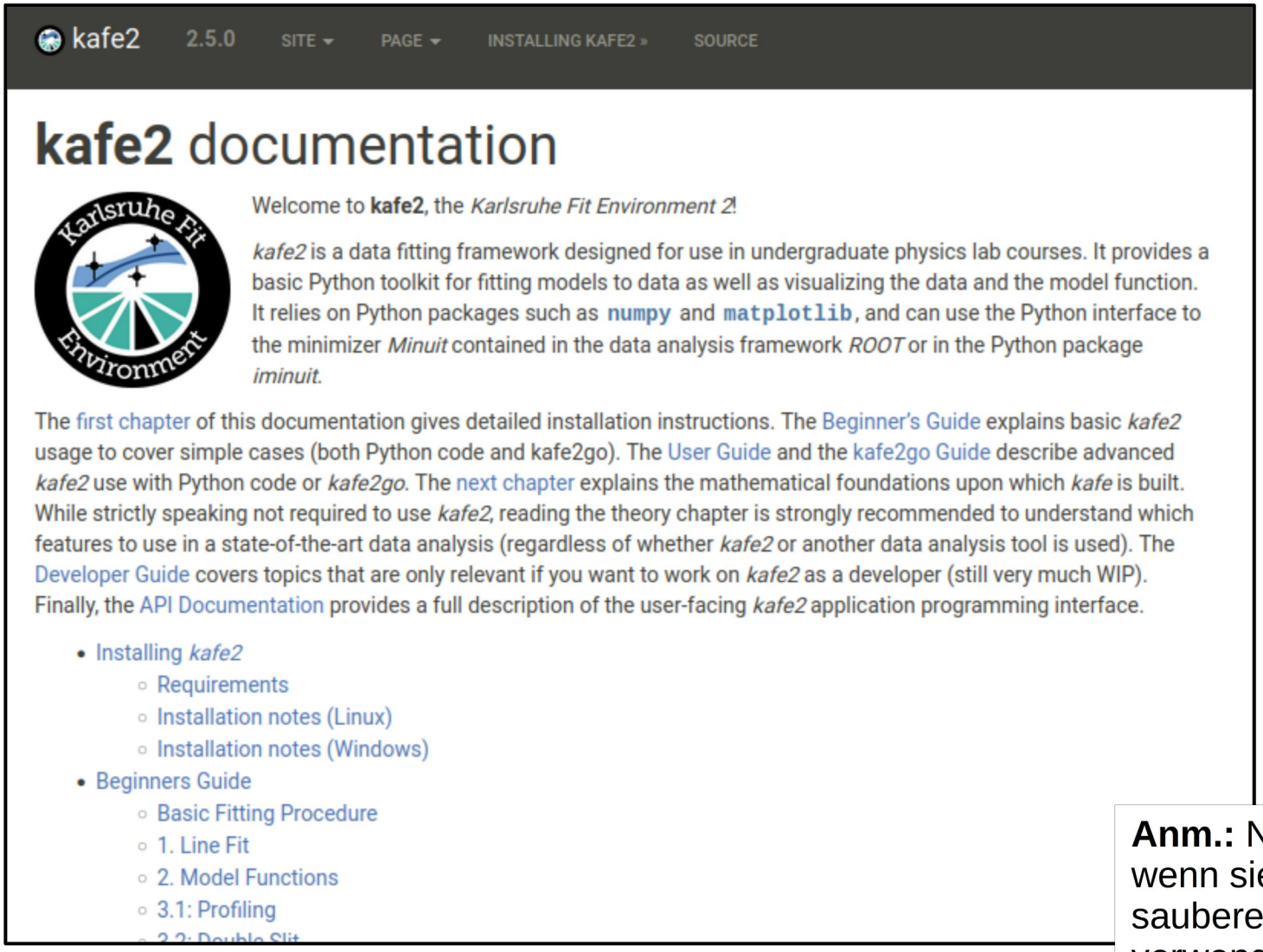
**Anm.:**

- In Ermangelung auch nur eines vernünftigen und einfach bedienbaren Werkzeugs zur statistisch sauberen (numerischen) Parameterbestimmung stellen wir Ihnen eigene Werkzeuge zur Verfügung, die Sie nutzen können und sollten.
- Machen Sie sich im Rahmen des P1 mit der Nutzung dieser Werkzeuge vertraut.
- Nehmen Sie sich die Zeit, es lohnt sich!

# Das Anpassungspaket kafe

Zu **kafe** gibt es eine Anzahl von gut dokumentierten Beispielen:

http://ekpwww.etp.kit.edu/~quast/kafe2/htmldoc



kafe2 2.5.0 SITE PAGE INSTALLING KAFE2 » SOURCE

## kafe2 documentation

Welcome to **kafe2**, the *Karlsruhe Fit Environment 2*!

*kafe2* is a data fitting framework designed for use in undergraduate physics lab courses. It provides a basic Python toolkit for fitting models to data as well as visualizing the data and the model function. It relies on Python packages such as `numpy` and `matplotlib`, and can use the Python interface to the minimizer *Minuit* contained in the data analysis framework *ROOT* or in the Python package *iminuit*.

The first chapter of this documentation gives detailed installation instructions. The Beginner's Guide explains basic *kafe2* usage to cover simple cases (both Python code and kafe2go). The User Guide and the kafe2go Guide describe advanced *kafe2* use with Python code or *kafe2go*. The next chapter explains the mathematical foundations upon which *kafe* is built. While strictly speaking not required to use *kafe2*, reading the theory chapter is strongly recommended to understand which features to use in a state-of-the-art data analysis (regardless of whether *kafe2* or another data analysis tool is used). The Developer Guide covers topics that are only relevant if you want to work on *kafe2* as a developer (still very much WIP). Finally, the API Documentation provides a full description of the user-facing *kafe2* application programming interface.

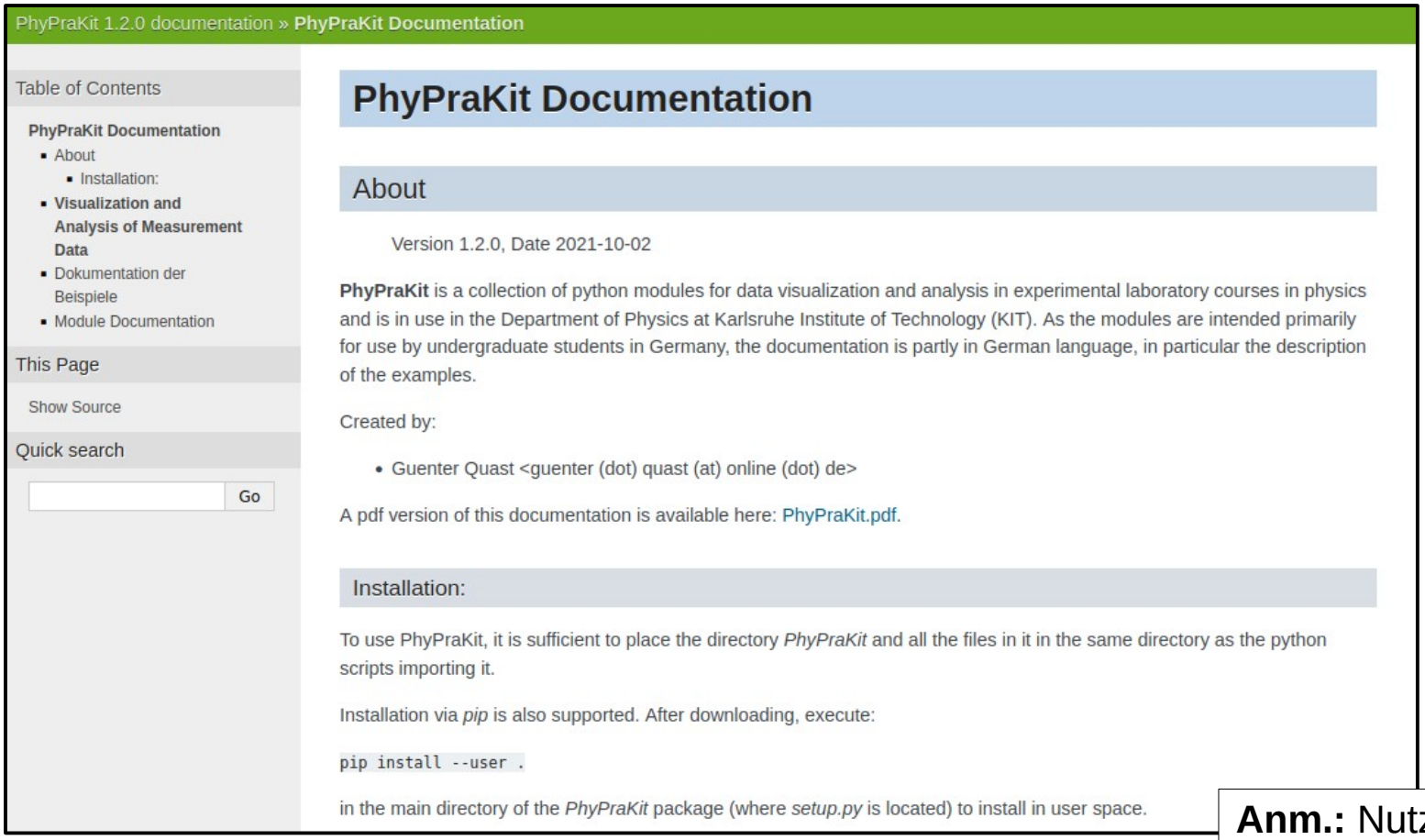
- Installing *kafe2*
  - Requirements
  - Installation notes (Linux)
  - Installation notes (Windows)
- Beginners Guide
  - Basic Fitting Procedure
  - 1. Line Fit
  - 2. Model Functions
  - 3.1: Profiling
  - 3.2: Double Slit

**Anm.:** Nutzen Sie kafe, wenn sie statisatisch sauberen state-of-the-art verwenden wollen.

# Das Anpassungspaket PhyPraKit

Bei **PhyPraKit** handelt es sich um einen wrapper um das in der Teilchenphysik etablierte Anpassungsprogramm minuit2

http://ekpwww.etp.kit.edu/~quast/PhyPraKit/htmldoc




**Anm.:** Nutzen Sie PhyPraKit, wenn Ihnen kafe zu kompliziert erscheint.

# Zusammenfassende Informationen…

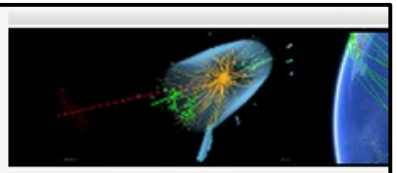
… erhalten Sie auf der Webseite von Prof. Günter Quast:

http://ekpwww.etp.kit.edu/~quast

Karlsruhe Institute of Technology

**Institut für Experimentelle Teilchenphysik**

Prof. Dr. Günter Quast



Adresse
Curriculum Vitae
Studierende
Research
Papers & Talks
Schule
IEKP
Links
Datenschutzerkärung

Vorlesung: Di., 12:00, Lehmann HS, Praktikum: Mo./Di., 14:00 - 18:00

- **Praktikum zur klassischen Physik**
  Übersicht zur computergestützten Datenauswertung im Praktikum
  Links zu Information und Materialien:
  - Vorlesung Fehlerrechnung
  - *jupyter* Notebook Tutorials zur Statistik und Fehlerrechnung im Praktikum -> Dateien zum Download
  - *python-Scripte* für das Praktikum zur klassischen Physik -> Downloadbereich
  - Software-Tools im Paket PhyPraKit, Download des gesamten Pakets mit Beispielen PhyPraKit-master.zip und Installationspaket für pip (.tar.gz)

**Skripte und Materialien**

- *jupyter* Notebook Tutorials zur Statistik und Fehlerrechnung im Praktikum Dateien zum Download
- **Software-Tools zu den physikalischen Praktika** PhyPraKit
  Download des gesamten Pakets mit Beispielen PhyPraKit-master.zip aktuelle Version auf github
  und Installationspaket für pip (.tar.gz)
- Computergestützte Datenauswertung
- **Funktionsanpassung** in Python, Paket **kafe2** (Dokumentation)
  *kafe2* auf GitHub Installationspaket kafe2 (whl) kafe2-master.zip

  Alte Version **kafe** Vers. 1. (Dokumentation)
  kafe Vers. 1 auf GitHub Installationspaket kafe (whl) kafe-master.zip
- Virtuelle Maschine zur Datenanalyse
- Docker-Container für Datenauswertung

Hier finden Sie große Mengen an nützlichen Links und Materialen, mit denen Sie sofort was anfangen können sollten.

# Empfohlene Software-Grundausstattung

Alle hier angegebenen software-Pakete sind quelloffen (Open Source), d.h. sie sind frei unter Linux, MS Windows, Mac OSX und auf dem CIP-Pool verfügbar

- Textdokumente erstellen mit LaTeX http://www.dante.de
- (Vektor-)Grafiken erstrellen mit inkscape https://inkscape.org/de
- Skript- und Programmiersprache python (vers. 3)
  unter Linux (meist) schon installiert
  für Windows siehe http://winpython.sourceforge.net
  (*Komplettpaket, enthält die unten angegebenen Zusatzpakete*)
  - Interaktive Umgebung für python: jupyter Notebooks
  - Grafische Darstellung mit python: matplotlib
  - Python-Bibliotheken für wissenschaftliches Arbeiten: numpy und scipy
- Funktionsanpassung mit python kafe2 http://ekpwww.etp.kit.edu/~quast/kafe2
- Tools für das Praktikum PhyPraKit http://ekpwww.etp.kit.edu/~quast/PhyPraKit

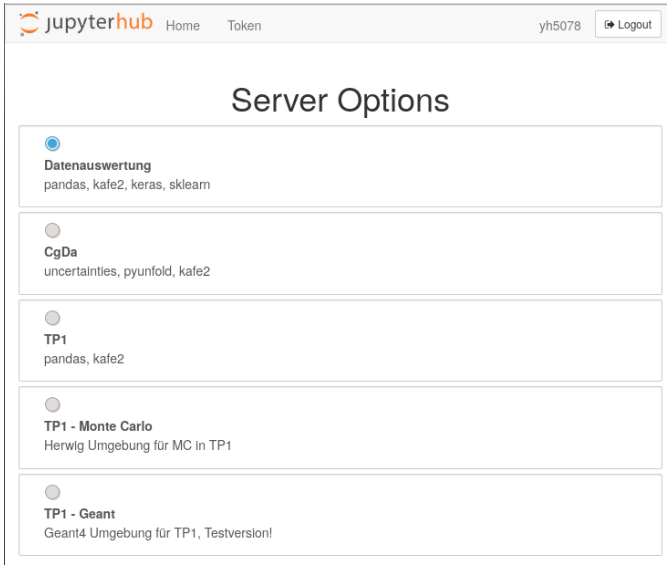
**Empfehlung: nutzen Sie *jupyter* Notebooks:** https://jupytermachine.etp.kit.edu
jupyter-Server der Fakultät für Physik

**Anleitung und Tutorials** http://ekpwww.etp.kit.edu/~quast/jupyter

Details s. VL CgDA https://comp.physik.kit.edu/Lehre/CGDA

# Arbeiten mit jupyter-Notebooks

https://jupytermachine.etp.kit.edu

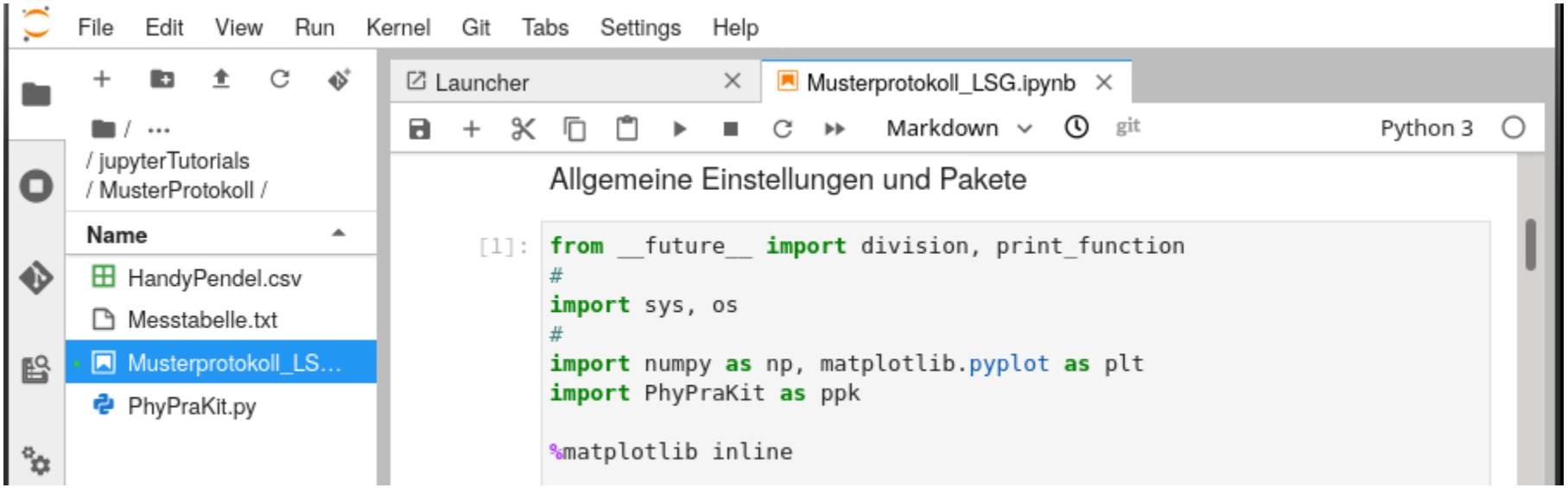



Voraussetzung zur **Nutzung des *jupyter*-Servers**:

- KIT-Account und CIP-Pool-Zugang
  s. https://comp.physik.kit.edu/Account/

**Beispiele:**

- Spickzettel für jupyter: JupyterCheatsheet.ipynb
- Einführung in python: PythonIntro.ipynb
- Spickzettel für *python*: PythonCheatsheet.ipynb
- Grundlagen der Statistik: IntroStatistik.ipynb
- Fehlerrechnung im Physikalischen Praktikum: Fehlerrechnung.ipynb
- Tutorium zu kafe2: kafe2Tutorial.ipynb



# Appell zum Schluss

**Sie sind Studierende der Physik:**

- Es ist für Physikstudierende nicht mehr zeitgemäß sich vor der Arbeit mit Computern zu „drücken“.
- Die Aufnahme und Verarbeitung von Daten ist Ihr Handwerk. Es wird Ihnen im P1 noch(!) wiederfahren, dass Sie Daten „händisch“ aufnehmen. Im Zeitalter von BigData wird dies für Sie jedoch immer weniger der Fall sein.
- Physikstudiernde sollten die (leicht zu erlerndende) Skript- und Programmiersprache **python** zumindest mal von weitem gesehen haben.
- Sie werden im Laufe Ihres Studiums mehr und mehr feststellen, dass seriös erfasste und verlässliche Unsicherheiten auf genommene Daten statistische Unsicherheiten sind.
- Wo immer Sie die Wahl haben, sei es durch die Art des Experimentierens oder die Planung des Experiments selbst, ersetzen Sie eigene Abschätzungen systematischer Unsicherheiten durch objektiv gewonnene statistische Unsicherheiten.
- Statistische Methoden auf Datensätzen stochastisch unabhängiger Ereignisse funktionieren immer!
- Ihr Anspruch an sich selbst sollte daher sein diese Methoden der Statistik zu erlernen und von grundauf zu verstehen.
- Ihr curriculum bietet Ihnen die die Möglichkeit(en) dazu.

# Anhang

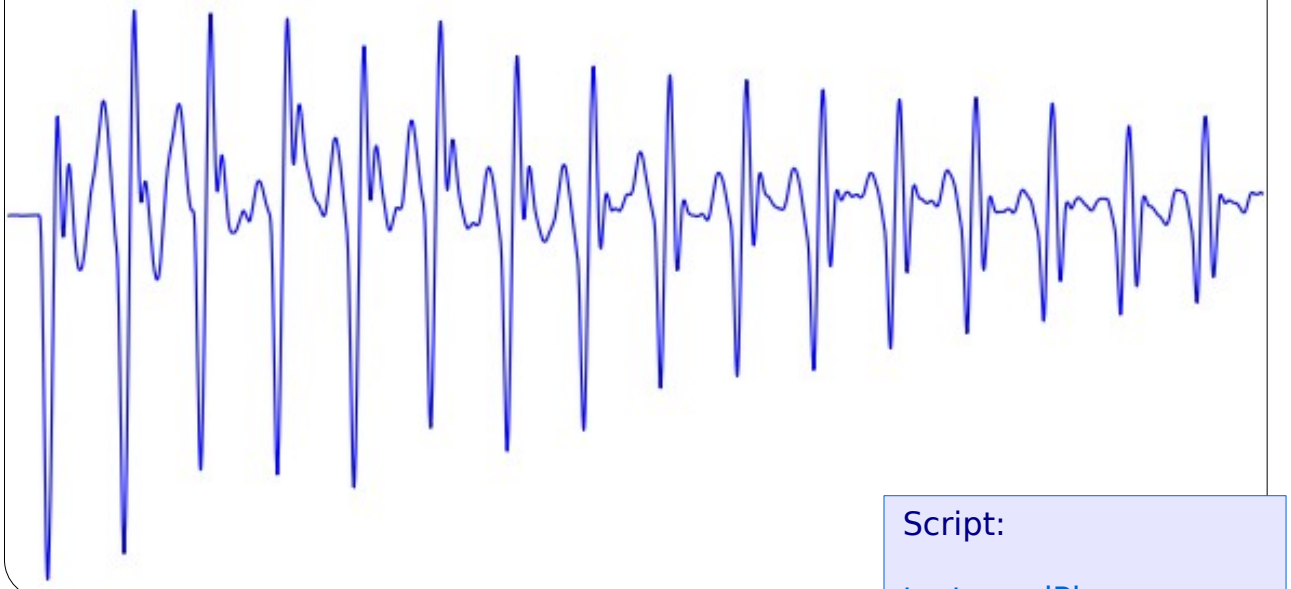
Praktische Beispiele

# Beispiel: Rohdaten einer Wellenform

```
Time,Channel A
(ms),(V)

-0.34927999,-0.00045778
-0.34799999,-0.00045778
-0.34671999,-0.00045778
-0.34543999,-0.00045778
-0.34415999,-0.00045778
-0.34287999,-0.00033570
-0.34159999,-0.00018311
-0.34031999,-0.00018311
-0.33903999,-0.00018311
-0.33775999,-0.00003052
-0.33647999,0.00006104 -
0.33519999,0.00006104 -
0.33391999,0.00006104 -
0.33263999,0.00006104 -
0.33135999,0.00021363 -
0.33007999,0.00021363

        . . .

9.63983995,0.02642903
9.64111995,0.02655110
9.64239995,0.02655110
9.64367995,0.02655110
9.64495995,0.02655110
9.64623995,0.02655110
9.64751995,0.02655110
9.64879995,0.02642903
9.65007995,0.02612384
9.65135995,0.02584918
9.65263995,0.02557451
9.65391995,0.02526933
9.65519995,0.02487259
```

7816 Wertepaare exportiert aus USB-Oszilloskop PicoScope im **csv**-Format (**c**omma **s**eparated **v**alues)

importiert in `numpy`-arrays `t` und `v`, dargestellt mit `plt.plot(t,v)`



Script:

test_readPicoscope.py

Datei

Welleenform.csv

# Beispiel Frequenzanalyse

## Amplitudenverlauf

$$a_i = \text{Amplitude}(t_i)$$

(„time domain“)

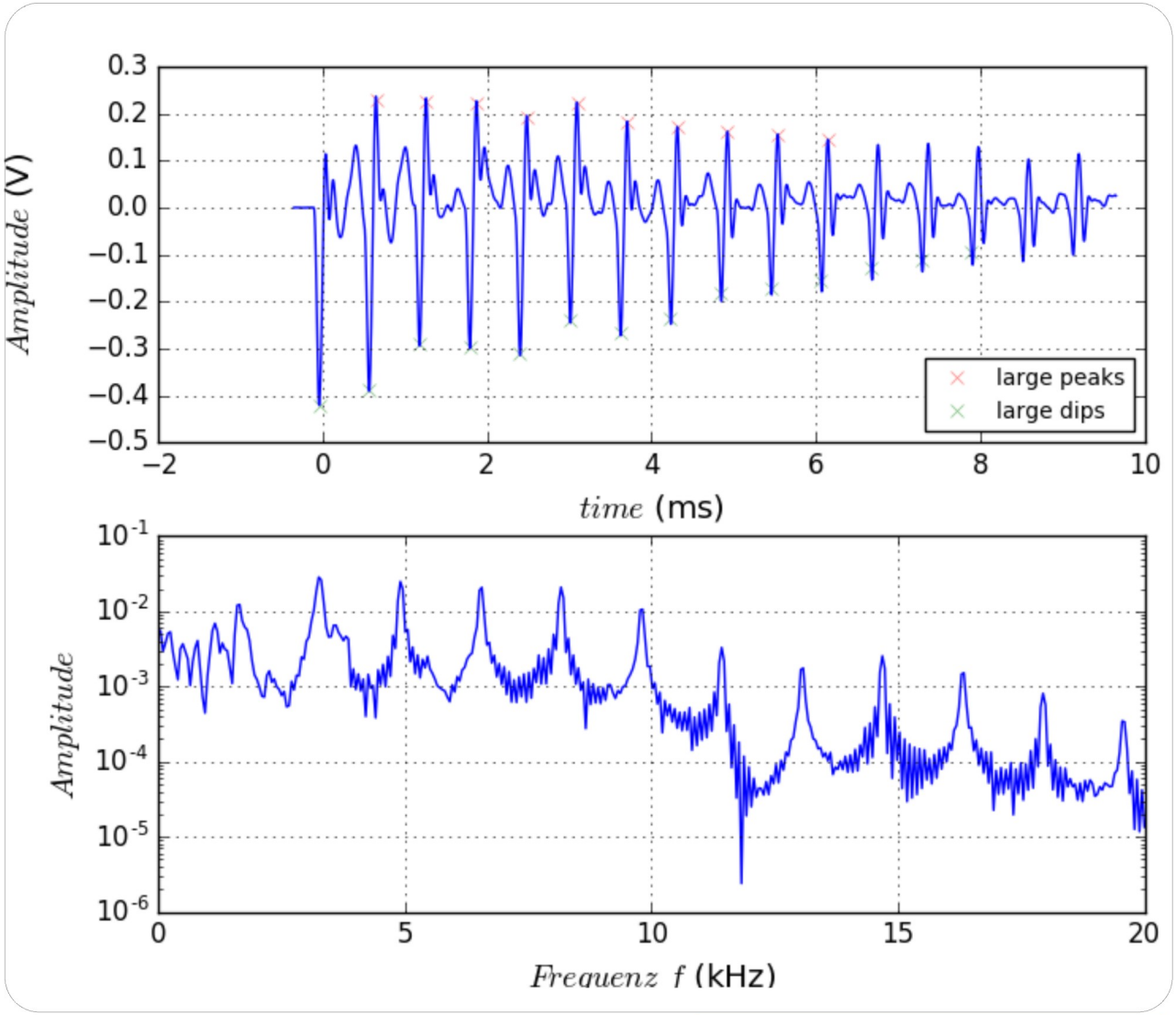
## Fourier-Transformation:

$$s_k = \frac{1}{n} \sum_i a_i \sin\left(\frac{2\pi}{f_k} t_i\right)$$

$$c_k = \frac{1}{n} \sum_i a_i \cos\left(\frac{2\pi}{f_k} t_i\right)$$

$$A_k = \sqrt{{s_k}^2 + {c_k}^2}$$

(„frequency domain“)




Skript `test_Fourier.py`

numerisch effizienteres Verfahren: **Fast Fourier Transform, FFT**

# Modellanpassung mit kafe

In der Praxis setzt man Programmpakete ein, die
- Strukturen zur Verwaltung von Daten und deren Fehlern
- Definition von Modellen
- Anpassung mittels numerischer Optimierung
- grafische Darstellung
- Ausgabe der Ergebnisse

bereit stellen.

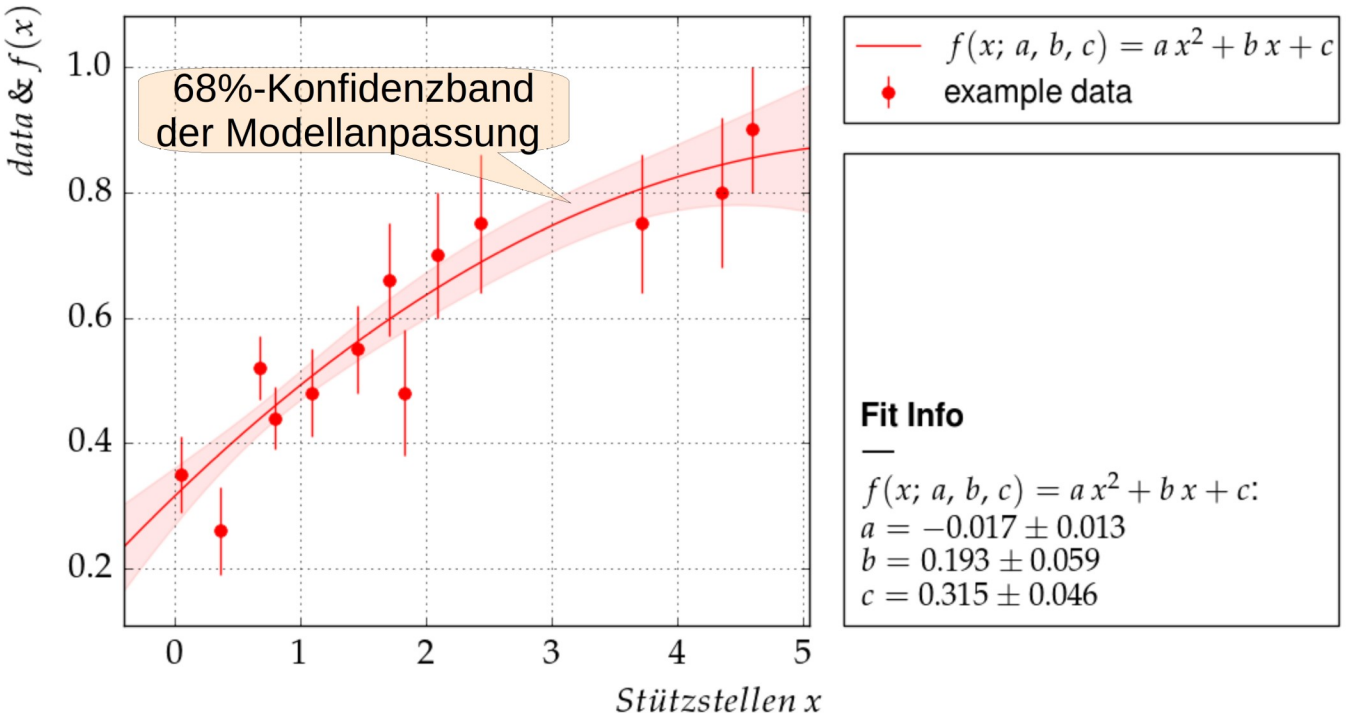
Zuverlässige Konvergenz zum globalen Minimum bei nicht-linearen Problemstellungen erfordert meist das Setzen geeigneter Startwerte!

Beispiel mit dem `python`-Framework

„**kafe**“:

Anpassung der drei Parameter einer quadratischen Funktion an Datenpunkte mit Unsicherheiten nur in Ordinatenrichtung.

siehe Skript
`fitexample_kafe.py`



# Beispielcode: numerische Anpassung mit kafe

siehe Script `fitexample_kafe.py`

```
# example fit with kafe
import kafe
from kafe.function_tools import FitFunction, LaTeX, ASCII

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

# --------- begin of workflow ----------------------
# set data
xm = [.05,0.36,0.68,0.80,1.09,1.46,1.71,1.83,2.44,2.09,3.72,4.36,4.60]
ym = [0.35,0.26,0.52,0.44,0.48,0.55,0.66,0.48,0.75,0.70,0.75,0.80,0.90]
ye = [0.06,0.07,0.05,0.05,0.07,0.07,0.09,0.1,0.11,0.1,0.11,0.12,0.1]
# create a kafe Dataset
kdata = kafe.Dataset(data=(xm, ym), basename='kData', title='example data')
kdata.add_error_source('y', 'simple', ye) # add uncertainties
kfit=kafe.Fit(kdata, poly2)  # create the Fit object from data & fit function
kfit.do_fit()                # perform fit
kplot=kafe.Plot(kfit)        # create plot object
kplot.axis_labels = [r'$St\"utzstellen \, x $', r'$data\,\&\,f(x)$']
kplot.plot_all()             # make plots
kplot.show()                 # show the plots
```

nur zur Verschönerung

der wichtige Code für kafe

`kafe` nutzt das CERN-Paket MINUIT zur numerischen Optimierung

# Modellanpassung: Daten-getriebene Anpassung

Die Methode kafe.file_tools.build_fit_from_file() ermöglicht es, einfache Anpassungen **vollständig über eine Datei** zu **steuern**:

Skript kafe_fit-from-file.py

```
import sys, matplotlib.pyplot as plt, kafe
from kafe.file_tools import buildFit_fromFile

# check for / read command line arguments
if len(sys.argv)==2:
  fname=sys.argv[1]
else:
  fname='data.fit'
print '*==* script ' + sys.argv[0]+ ' executing \n',\
      '     processing file ' + fname

# initialize fit object from file and run fit
theFit = buildFit_fromFile(fname)
theFit.do_fit()
thePlot = kafe.Plot(theFit)
thePlot.plot_all( show_info_for=None)

#thePlot.save(fname.split('.')[0]+'.pdf')
#theFit.plot_correlations() # eventually contours
thePlot.show() # show everything on screen
```

Datei IU-Messungen.fit

```
# example showing fit driven by input file

# – - Meta data for plotting
*BASENAME linearFitExample
*FITLABEL   Angepasste Gerade
*TITLE  I-U Messungen
*xLabel $U$
*xUnit   V
*yLabel $I$
*yUnit   A

# - - description of data
*xData
0.30   0.01
  . . .
0.90   0.05
*yData
0.25   0.01
   . . .
0.88   0.02
# systematic errors: correlated among all
measurements
*xRelCor 0.005   # relative error of 0.5% on U
*yRelCor 0.005   # relative error of 0.5% on I

# - - fuction to fit
*FitFunction
def fitf(x, a=1., b=0.):
~~return a*x +b

 # initial parameter values and range
*InitialParameters
1.   0.3
0.   0.1
```

Mit Hilfe dieses Universalskripts müssen Sie keinen speziellen pyhton-Code schreiben, sondern nur die Eingabedatei erstellen !

# Kommandozeilen-Werkzeug: kfitf.py

Es gibt ein Werkzeug für die Kommandozeile, **kfitf.py**
das in einer .fit -Datei definierte Anpassung ausführt:

```
kfitf.py
usage: kfit [-h] [-n] [-s] [-c] [--noinfo] [--noband] [-f FORMAT] filename

Perform a fit with the kafe package driven by input file

positional arguments:
 filename              name of fit input file

optional arguments:
   -h, --help          show this help message and exit
   -n, --noplot        suppress ouput of plots on screen
   -s, --saveplot      save plot(s) in file(s)
   -c, --contour       plot contours and profiles
   --noinfo            suppress fit info on plot
   --noband            suppress 1-sigma band around function
   -f FORMAT, --format FORMAT
                       graphics output format, default=pdf
```

# weitere Beispiele ...

Alle Dateien zu den gezeigten Beispielen (und noch viele mehr) finden Sie unter dem Link zur Vorlesung

**Computergestützte Datenauswertung**

http://www.ekp.kit.edu/~quast/CgDA/CgDA.html

Spezielle Python-Beispiele zum Start ins Praktikum

http://www.ekp.kit.edu/~quast/CgDA/PhysPrakt

## das `python-Modul PhyPraKit.py` enthält

1. Einlesen von Daten aus Text-Dateien
   - read_data() n Spalten mit Meta-Daten (Datum, Autor, …)
   - read_csv() Daten als „Comma Separated Values“
   - read_txt() allg. Daten im Text-Format
2. Methoden zur Prozessierung von Roh-Daten
   Glätten, Suche nach Extrema und Flanken, Fouriertransformation, Autokorrelation
3. Berechnung des gewichteten Mittelwerts
4. Histogramm-Tools
   - barstat() statistische Information aus Balkendiagramm
   - histstat() statistiche information aus 1d-Histogramm
   - hist2dstat() statistische Information aus 2d-histogram
   - profile2d() "profile plot" für 2d-Daten
   - chi2p_indep2d() chi2-Test auf Unabhängigkeit zweier Datensätze
5. lineare Regression `(y = a · x + b)`, Funktionsanpassung
   - linRegression() mit der analytischen Formel (nur y-Fehler)
   - kFit() numerisch mit `kafe`, x-, y- und korrelierte Fehler
6. Erzeugung von simulierten Datensätzen

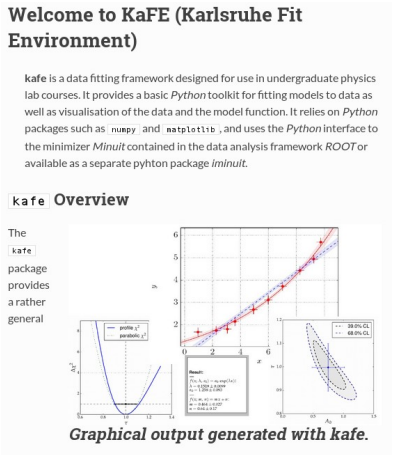
s. Beispiel-Scripts

```
test_readtxt.py

test_Fourier
test_AutoCorrelation.py

test_Histogram.py

test_linRegression.py
test_kFit.py

test_generateData.py
```

# Literatur

- M. Erdmann, T. Hebbeker, A. Schmidt, *Statistische Methoden in der Experimentalphysik, Pearson, 2019*
- H.J.Eichler et al., *Das neue Physikalische Grundpraktikum*, Springer Lehrbuch 2016 *digital über die KIT-Bibliothek*
- W.H. Gränicher; *Messung beendet - was nun?,* Teubnerverlag Stuttgart, 1996
- Gerhard Bohm, Günter Zech; *Introduction to Statistics and Data Analysis for Physicists*, Verlag Deutsches Elektronen-Synchrotron 2014 *digital als DESY ebook*
- Eigene Skripte zur Veranstaltung
  - Funktionsanpassung
  - Software-Paket *kafe*
  - *jupyter* Server
  - Virtuelle Maschine

http://www.ekp.kit.edu/~quast


**Welcome to KaFE (Karlsruhe Fit Environment)**

**kafe** is a data fitting framework designed for use in undergraduate physics lab courses. It provides a basic *Python* toolkit for fitting models to data as well as visualisation of the data and the model function. It relies on *Python* packages such as `numpy` and `matplotlib`, and uses the *Python* interface to the minimizer *Minuit* contained in the data analysis framework *ROOT* or available as a separate pyhton package *iminuit*.

`kafe` **Overview**

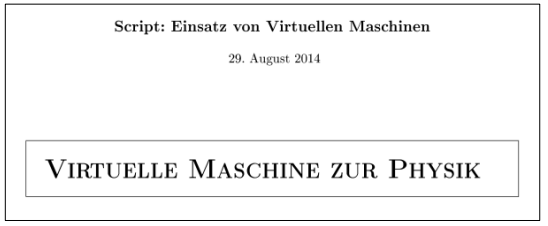
The `kafe` package provides a rather general

***Graphical output generated with kafe.***

Script: Anpassen von Funktionen an Messdaten

26. Februar 2015

Funktionsanpassung mit der $\chi^2$-Methode


Script: Einsatz von Virtuellen Maschinen

29. August 2014

VIRTUELLE MASCHINE ZUR PHYSIK

**Danke für Ihre Aufmerksamkeit**

Wir wünschen Ihnen

Experimentierfreude und ein

spannendes, erkenntnisreiches Praktikum