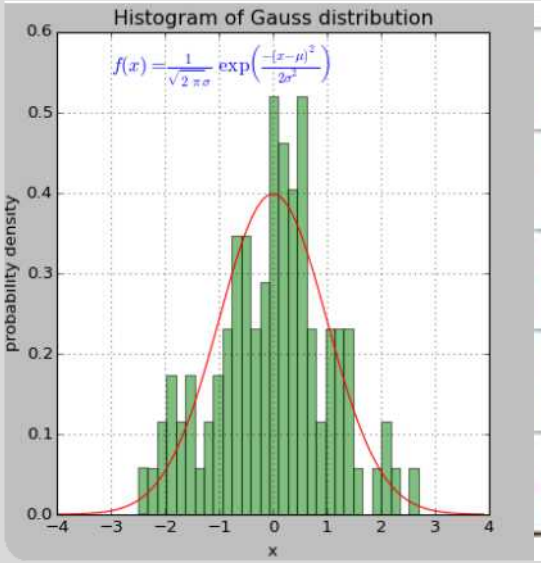
Karlsruher Institut für Technologie

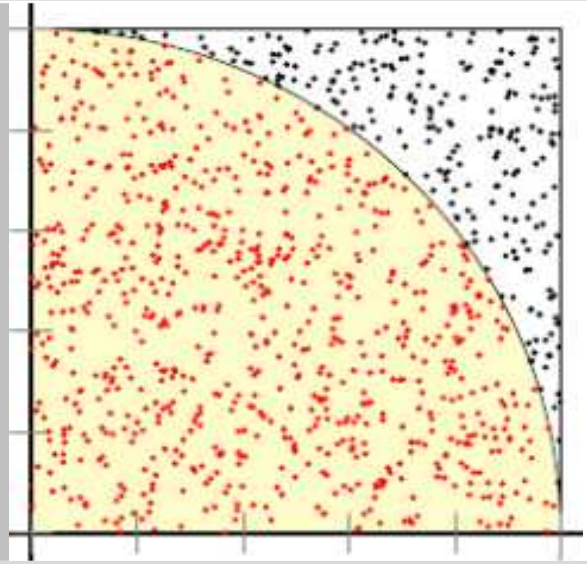
Zusammenfassung und Konzept
Computergestützte Datenauswertung

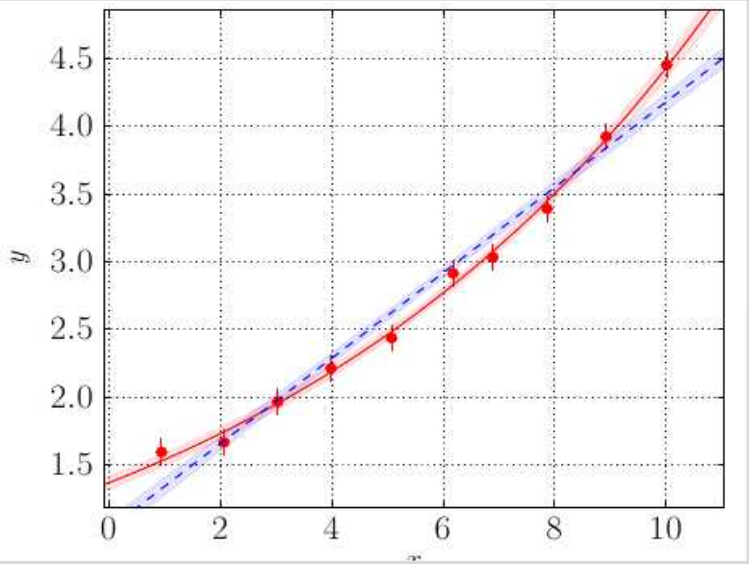
Günter Quast, Andreas Poenicke

Fakultät für Physik
Institut für Experimentelle Kernphysik

SS 2016







Neue Veranstaltung

„Rechnergestützte Datenauswertung mit Computerübung“

mit 1 SWS Vorlesung und 1 SWS Übung erstmalig im SS16

(als verpflichtender Kurs Schlüsselkompetenz, 2. Semester im Bachelor Physik)

G. Quast, A. Poenicke & viele Assistenten

Ziele:

– einheitliche Arbeitsumgebung für alle Studierenden
(für Windows, Linux & Mac OS, CIP-Pool, basierend auf `python/matplotlib`)

– grafische Darstellung von Funktionen und Daten

– Anpassung von Funktionen (=Modellen) an Messdaten
& grundsätzliches Verständnis des theoretischen Hintergrunds

– Modellierung von experimentellen Daten („Monte Carlo“)

– Beherrschung des `python`-Pakets `kafe` zur Funktionsanpassung

Inhalt der Vorlesung

- Einführung, Umgang mit dem Computer,
  Aufsetzen der eigenen Arbeitsumgebung
  - vorzugsweise unter dem Betriebssystem Linux
  - oder in einer „virtuellen Maschine“,
  - alternativ unter MS Windows 10 oder MAC OSX
- Eingabe und Darstellung von Daten mit *python / matplotlib*
- Statistische Grundlagen: diskrete und kontinuierliche Verteilungen, Erwartungswert und Varianz/Standardabweichung, Fehlerfortpflanzung,
- Die Rolle des Zufalls: künstliche“ Daten mit der MC-Methode
- Bearbeitung digitaler Daten: Signalverarbeitung
- Parameterschätzung : Methode der kleinsten Quadrate, Parameterunsicherheiten, Likelihood
- Software: Beispiele mit kafe

Inhalt der Übungen

1.1: Aufsetzen der Arbeitsumgebung
1.2: Kennenlernen der Arbeitsumgebung
1.3: Kennenlernen von `Python`

2.1: Kennenlernen von `Python` (2)
2.2: Arbeiten mit `numpy`
2.2: Arbeiten mit `matplotlib`

3.1: Darstellen von Funktionen
3.2: Berechnung statistischer Größen
3.3: Funktionen von Zufallszahlen

4.1: Würfelspiel
4.2: Zentraler Grenzwertsatz
4.3: Parameterschätzung: Resonanzkurve mit Messdaten

5.1 Signalverarbeitung: Frequenzbestimmung
5.2: Korrelation
5.3: Anpassung von Modellen an Daten mit `kafe`

6: Datenauswertung
Frequenzbestimmung beim Federpendel und
Messung der Erdbeschleunigung

Empfohlene Software-Grundausstattung

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

- Textdokumente erstellen mit LaTeX — siehe z.B. http://www.dante.de/
- (Vektor-)Grafik mit inkscape — https://inkscape.org/de/
- Ausführen von virtuellen Maschinen: VirtualBox — https://www.virtualbox.org/
- Script- und (Programmier-) Sprache python vers. 2.7
  unter Linux (meist) schon installiert
  für Windows siehe — http://winpython.sourceforge.net/
  (Komplettpaket, enthält schon die unten angegebenen Zusatzpakete)

  (einfache) Entwicklungsumgebung für python IDLE — **I**ntegrated **D**velopment and **LE**arning

  interaktives Arbeiten mit python: IPython — http://ipython.org/notebook.html

  grafische Darstellung mit python: matplotlib

  python-Bibliotheken für wissenschaftliches Arbeiten: numpy und scipy
- Funktionsanpassung mit python: kafe — http://www.ekp.kit.edu/~quast
- *man braucht noch einen* Texteditor *(das ist „Geschmackssache“ ,*
  etwas komfortabler als MS Notepad *darf's schon sein)*

Details s. CgDA, V01b_intro.pdf http://www.ekp.kit.edu/~quast/CgDA (**user:** Students, **password:** only)

enthält auch Hinweise und ein vorkonfiguriertes Gesamtpaket für Win10, sowie eine „virtuelle Maschine“

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. Dateneingabe aus Text-Dateien
2. Signalprozessierung (Filter, Fourier-Transf., Autokorrelation)
3. Berechnung des gewichteten Mittelwerts
3. 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 Unabhänggkeit zweier Datensätze
4. (lineare) Regression
   kRegression() mit `kafe`, x-, y- und and korrelierte Fehler
5. Erzeugung von simulierten Datensätzen
   smearData()
   generateData()

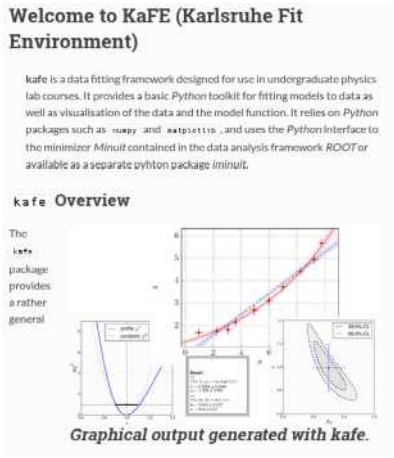
s. Beispiel-Scripts

```
test_readColumnData.py
test_readCSV.py
test_readtxt.py
test_convolutionFilter.py
test_autoCorrelation.py
test_histogram.py
test_linRegression.py
test_kRegression.py
test_generateData.py
```

Literatur

- Eigene Skripte zur Veranstaltung
  - Funktionsanpassung
  - Software-Paket *kafe*
  - 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

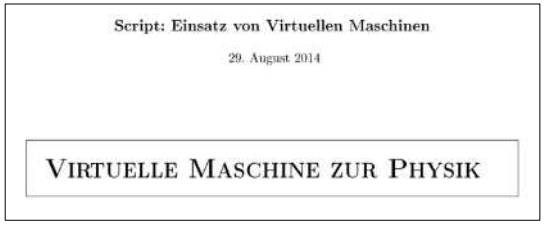
The kafe package provides a rather general

Graphical output generated with kafe.


Script: Anpassen von Funktionen an Messdaten

26. Februar 2015

Funktionsanpassung mit der χ^2-Methode

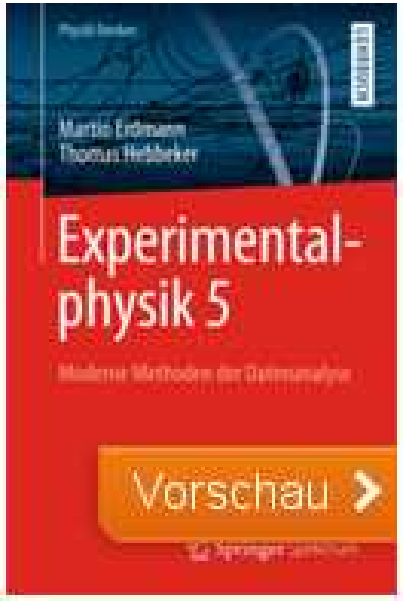

Script: Einsatz von Virtuellen Maschinen

29. August 2014

VIRTUELLE MASCHINE ZUR PHYSIK

- M. Erdmann und T. Hebbeker, Experimentalphysik 5
  Moderne Methoden der Datenanalyse
  Springer Lehrbuch 2013

digital über die KIT-Bibliothek



Überblick über ausgewählte Inhalte

Softwareumgebung

& friends:
- numpy
- scipy
- matplotlib

python ist eine
– moderne
– objektorientierte
– interpretierte
– intuitive
– leicht zu erlernende
– effiziente
Skript- und Programmier- Sprache

mit
– *Unterstützung praktisch aller Funktionen des Betriebssystems,*
– *eigenen Modulen insb. auch für wissenschaftliches Rechnen,*
– *Anbindung von Bibliotheken wichtiger Programmpakete,*
– *Einbindung von selbst-programmierter Funktionalität.*

python ist verfügbar für Windows, Mac und Linux: *Platform-übergreifend !*

Wozu Statistische Datenanalyse ?

Die Aufgabe des Wissenschaftlers:

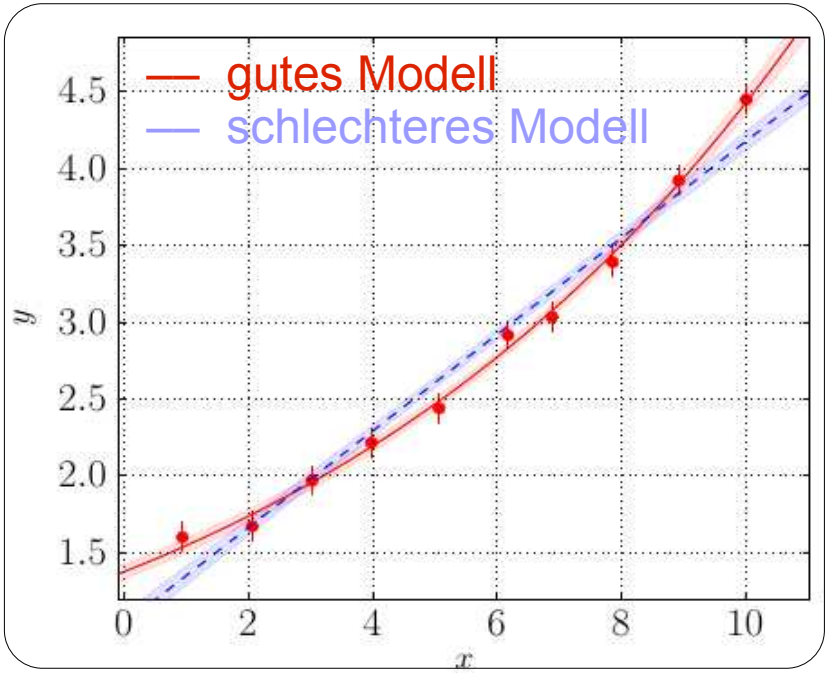
Vergleich von Modellen mit der Wirklichkeit

Frage:
Passt das (theoretische) Modell zu den Beobachtungen (Messungen) ?

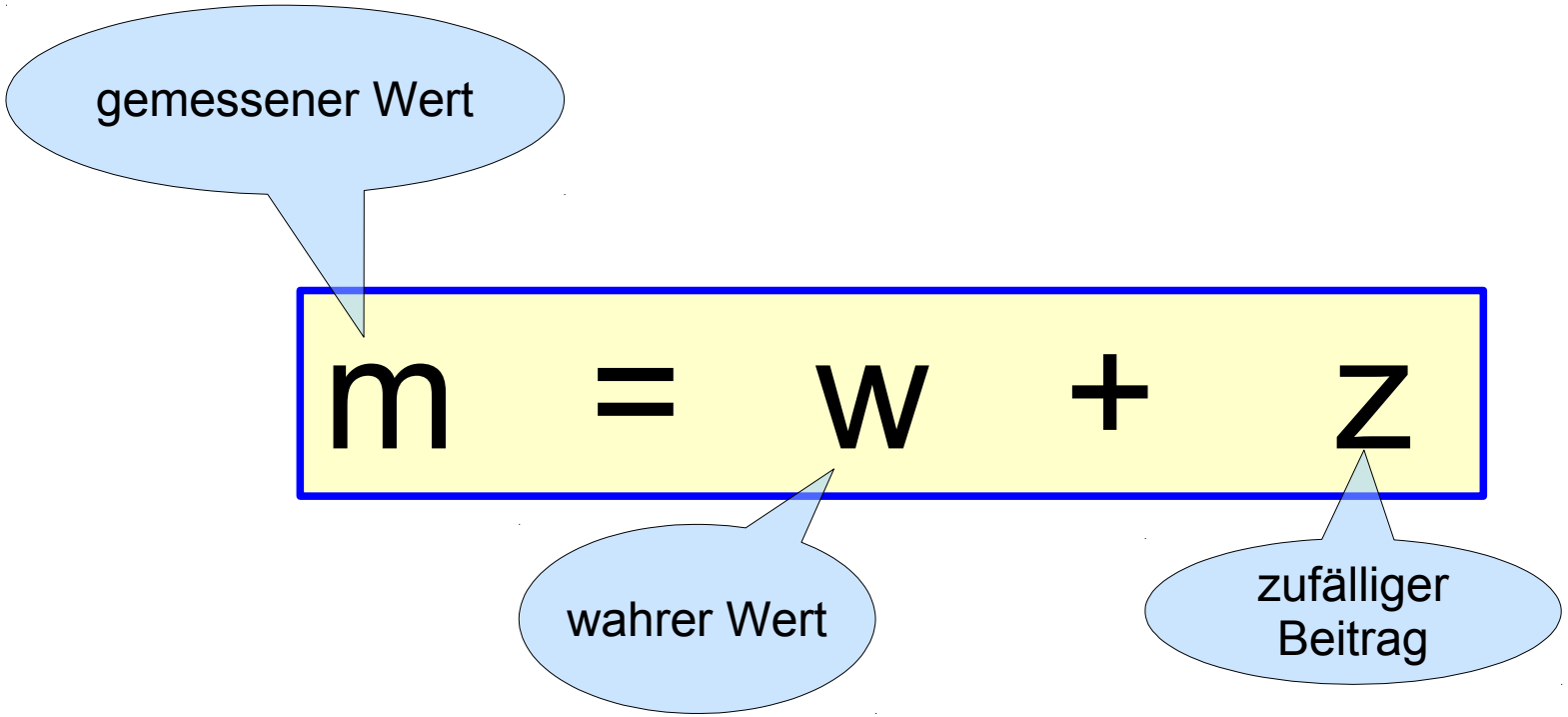
- wenn nein: Modell verwerfen oder verbessern
- wenn ja: freie Modellparameter bestimmen

– **Messungen sind immer mit statistischen Ungenauigkeiten behaftet**
(Rauschen, Ablese- oder Digitalisierungs-genauigteit, Eichung, ...)

– **in der Quantenphysik sind die relevanten Modellvorhersagen und -Größen selbst Parameter von Zufallsverteilungen**
(mittlere Lebensdauer eines Zustands, Erwartungswert eines Operators, ...)



Modell einer Messung

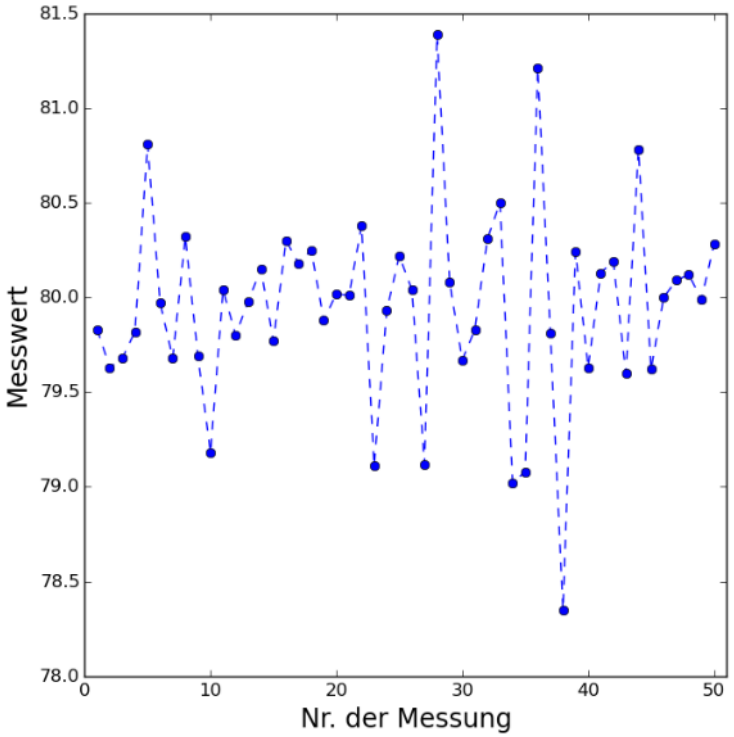


***z* kann viele Ursachen haben:**

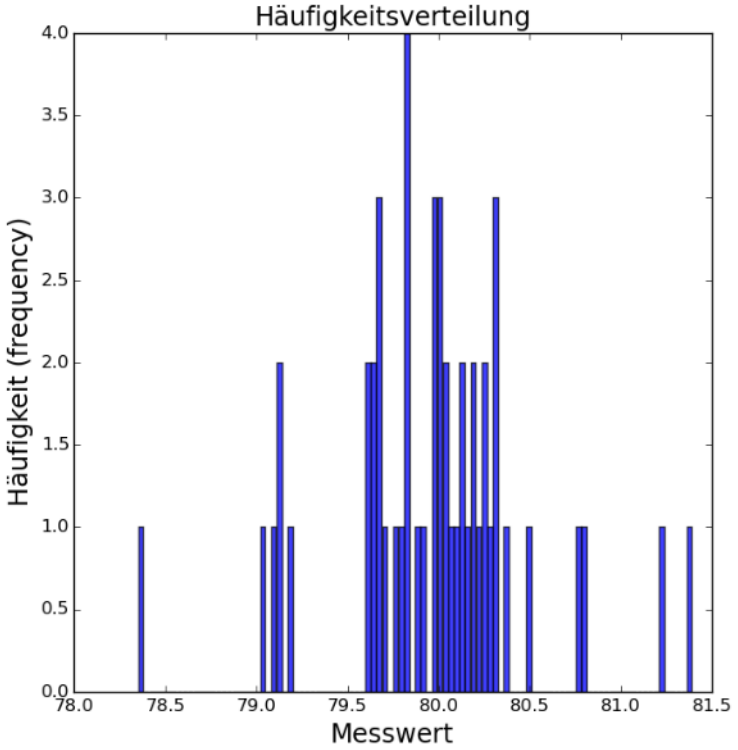
- zufälliger Beitrag zum Messwert („Rauschen“) → „statistische Unsicherheit“
- Genauigkeit des verwendeten Messinstruments → „systematische Unsicherheit“
- mitunter gibt es auch eine Unsicherheit auf den „wahren“ Wert, den man oft z zuschlägt → „theoretische Unsicherheit“
- Fehler im Messprozess – sollten nicht passieren !

$$\Rightarrow \quad z = z_{stat} + z_{syst} + z_{theo}$$

Beispiel Längenmessung



Visualisierung der Messreihe
(typisch wie z.B. mit MS Excel)



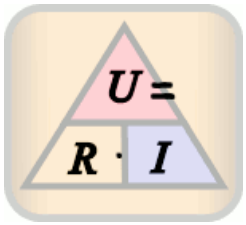
Darstellung als
„**Häufigkeitsverteilung**“
oder „**Histogramm**“
(*engl.* frequency distribution“, „histogram“)

Statistische Information: →

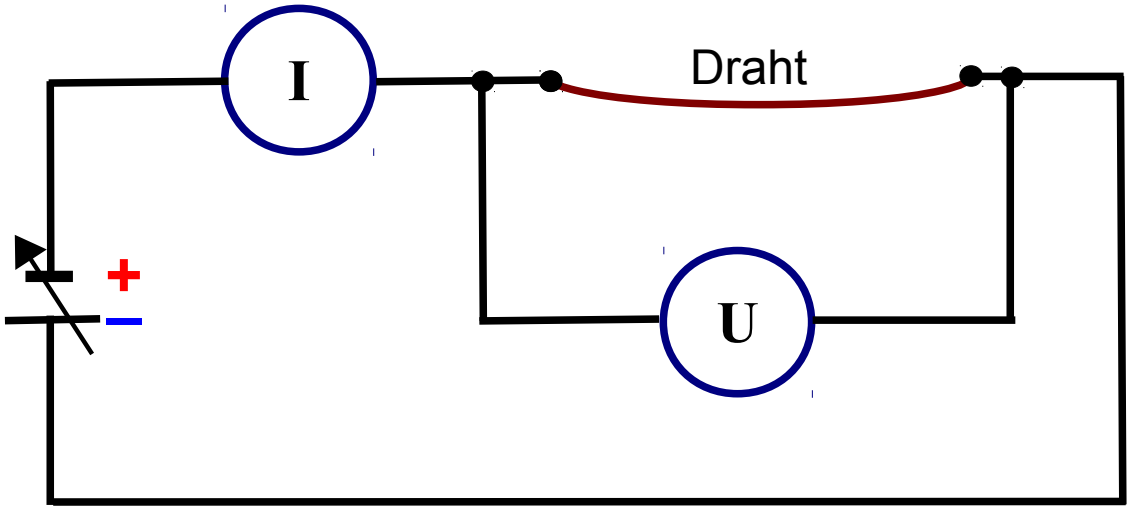
Mittelwert: 79.954
Standardabweichung: 0.522
Unsicherheit auf Mittelwert: 0.074

Anm.: Standardabweichung beschreibt „Breite der Verteilung“

Beispiel Strom-Spannungsmessung (1)



?



Messreihe:

U[V]	I[A]
0.15	0.14
0.25	0.22
0.40	0.36
0.50	0.48
0.60	0.53
0.70	0.64
0.75	0.65
0.80	0.74
0.90	0.81

Grafische Darstellung mit
(Millimeter -) Papier und Bleistift
oder Tabellenkalkulation beherrschen Sie alle

mit ein wenig Python-Code wird es komfortabler ...

Beispiel Strom-Spannungsmessung (2)

Messreihe:

U[V]	I[A]
0.15	0.14
0.25	0.22
0.40	0.36
0.50	0.48
0.60	0.53
0.70	0.64
0.75	0.65
0.80	0.74
0.90	0.81

Python Code

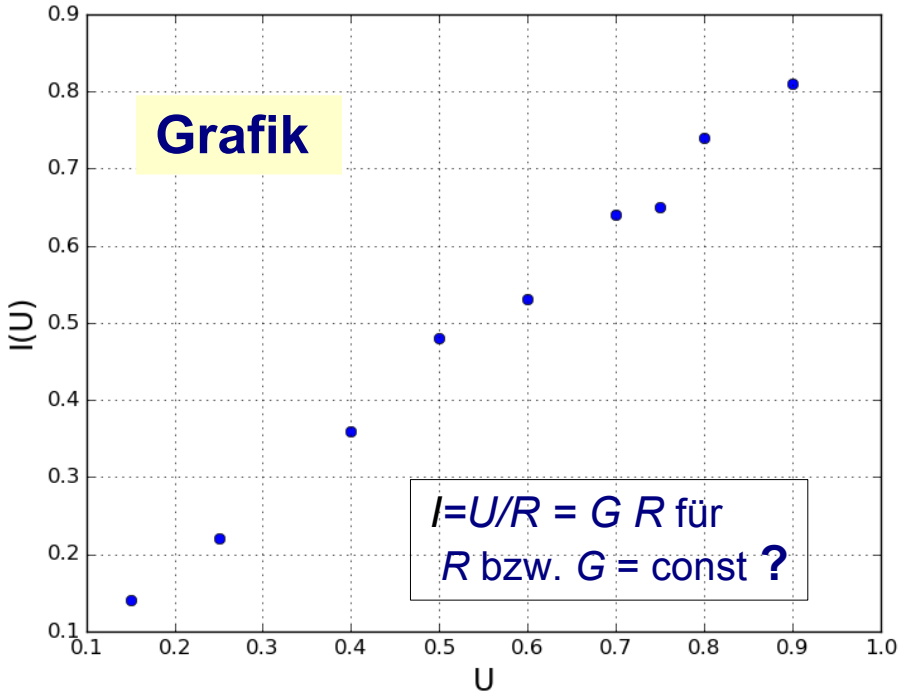
```
import numpy as np
import matplotlib.pyplot as plt

#read columns from file
infile="DataOhmsLaw.dat"
U, I = np.loadtxt(infile,unpack=True)

# plot the results
fig, ax=plt.subplots(1,1)
ax.errorbar(U, I, fmt='o')
ax.set_xlabel('U')
ax.set_ylabel('I(U)')
plt.show()
```

Frage:
Gilt das Ohm'sche Gesetz ?

```
# Measurements: Ohm's Law
#  Prec.: U: +\-0.015V, I: +\-0.015A
#  U[V]  I[A]
  0.15  0.14
  0.25  0.22
  0.40  0.36
  0.50  0.48
  0.60  0.53
  0.70  0.64
  0.75  0.65
  0.80  0.74
  0.90  0.81
```

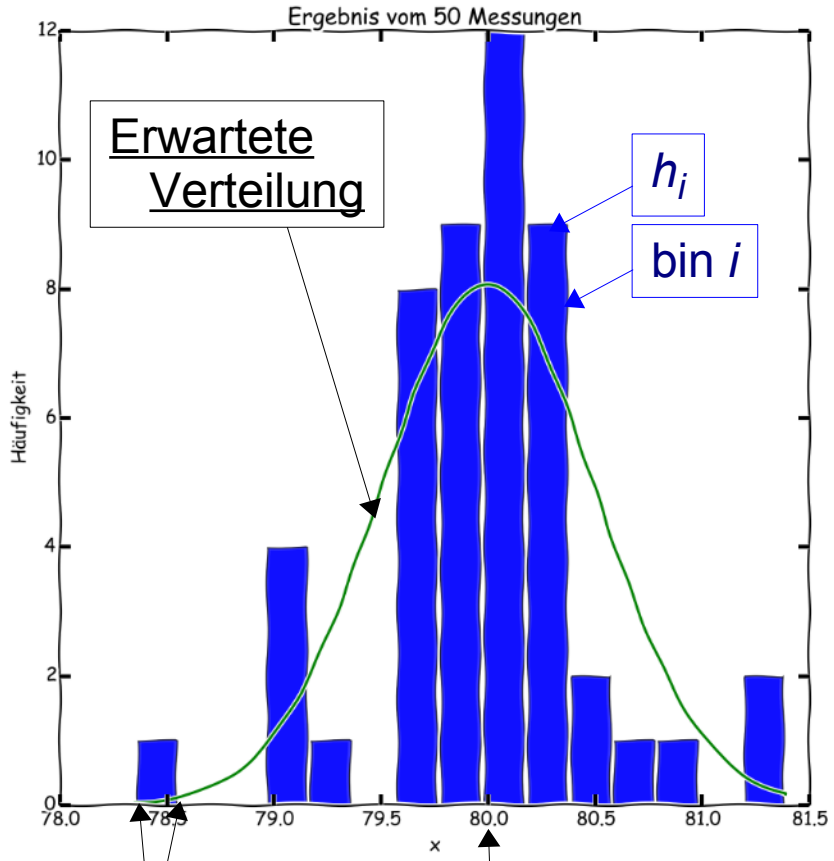
Text-Datei
DataOhmsLaw.dat

Grafik



Häufigkeitsverteilung und Wahrscheinlichkeitsdichte

Beispiel **Messen**:
Häufigkeit von N Messergebnissen in bestimmten Intervallen („Bins“)



Erwartete Verteilung

h_i

bin *i*

Für eine große Anzahl von Messungen nähert sich die Häufigkeitsverteilung der erwarteten Verteilung immer mehr an

Intervallgrenzen („bin edges“)

Erwartungswert der Verteilung

Häufigkeitsdefinition der Wahrscheinlichkeit:

$$p_i = h_i/N, \;\; N \to \infty$$

Charakterisierung v. Verteilungen: Formeln

	Stichprobe	diskrete Verteilung	kontinuierliche Vert.
Erwartungswert $\mathrm{E}[x] \equiv \langle x \rangle = \mu$	$= \frac{1}{N} \sum_{i=1}^{N} x_i$	$= \sum_{i=1}^{N} x_i p(x_i)$	$= \int_{-\infty}^{\infty} x f(x)\, dx$
Varianz $V \equiv \sigma^2 = \mathrm{E}\left[(x-\mu)^2\right] = \mathrm{E}\left[x^2\right] - \mu^2$	$= \frac{1}{(N-1)}{}^{*} \sum_{i=1}^{N} {x_i}^2 - \mu^2$	$= \sum_{i=1}^{N} {x_i}^2 p(x_i) - \mu^2$	$= \int_{-\infty}^{\infty} x^2 f(x)\, dx - \mu^2$

Schiefe

$\gamma_1 = \mathrm{E}\left[(x-\mu)^3\right] / \sigma^3$

Kurtosis

$\gamma_2 = \beta_2 - 3$ mit

$\beta_2 = \mathrm{E}\left[(x-\mu)^4\right] / \sigma^4$

$\gamma_2 = 0$ für Gauß-Vert.

+ höhere ...

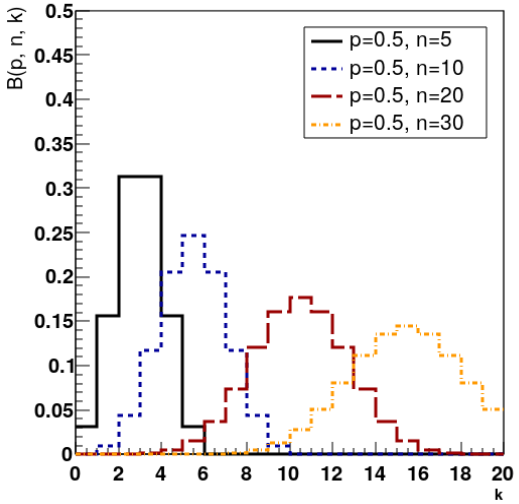
* „Bessel-Korrektur“: N → N-1 vermeidet Verzerrung
Plausibilitätsargument: für N=1 ist V nicht definiert !

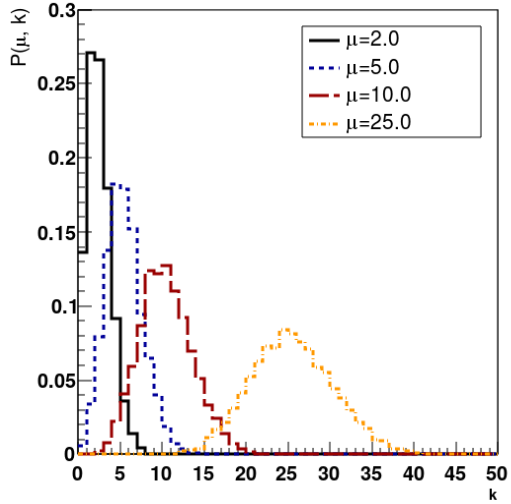
$$\mathrm{E}(x^n) = \sum_{i=1}^{N} {x_i}^n p(x_i) \text{ bzw. } \int_{-\infty}^{\infty} x^n f(x)\, dx$$

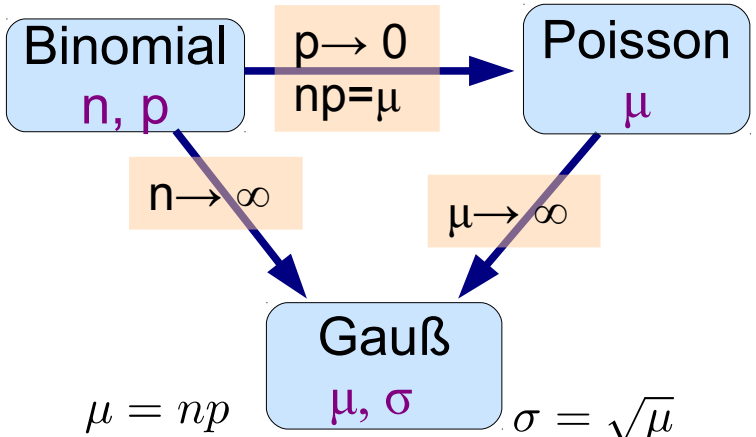
nennt man das n-te Moment der Verteilung

Eine Verteilung ist über ihre Momente vollständig charakterisiert

Gauss ↔ Poisson ↔ Binomial - Verteilung

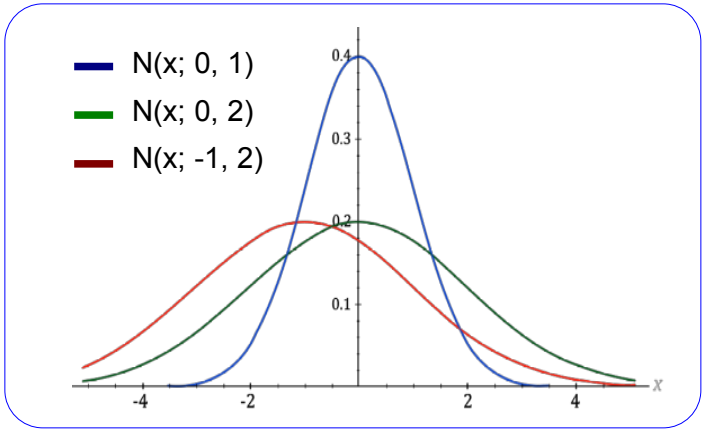






$$\mu = np \qquad \sigma = \sqrt{\mu}$$

$$\sigma = \sqrt{np(1-p)}$$



Zentraler Grenzwertsatz oder warum sind Messfehler Gauß-verteilt ?

Im Grenzfall von großen N ist die Summe von N unabhängigen Zufallszahlen eine Zufallszahl, die einer Gauß-Verteilung folgt.

$$x = \lim_{N\to\infty} \sum_{i=1}^{N} x_i$$

x_i aus beliebiger Verteilung mit Mittelwert μ_i und **endlicher Varianz** σ_i

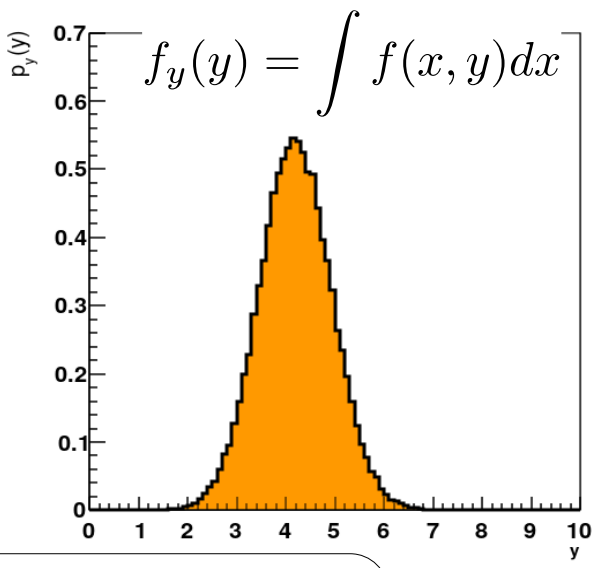
unter recht schwachen Annahmen (Lyapunov-Bedingung) **gilt:**

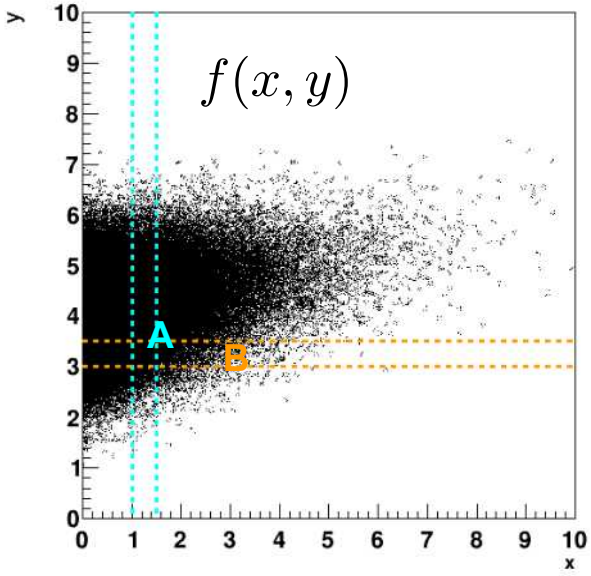
x **ist gaußverteilt mit Erwartungswert** $\mu = \sum_{i=1}^{N} \mu_i$ **und Varianz** $\sigma^2 = \sum_{i=1}^{N} {\sigma_i}^2$

Mehrdimensionale Verteilungen

Normierung

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} p(x,y)\mathrm{d}x\mathrm{d}y = 1$$





Projektion auf x-Achse

Projektion auf y-Achse

kumulativ

$$\mathcal{P}(x' \le x, y' \le y) = \int_{-\infty}^{x}\int_{-\infty}^{y} p(x',y')\mathrm{d}x'\mathrm{d}y'$$

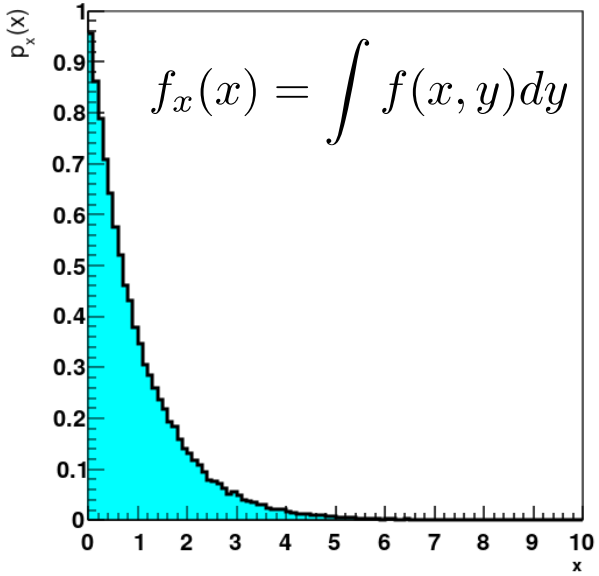
$$f_x(x) = \int_{-\infty}^{\infty} f(x,y)\mathrm{d}y$$

$$\mathrm{f}_y(y) = \int_{-\infty}^{\infty} f(x,y)\mathrm{d}x$$

Randverteilungen
engl.: ***marginal distribution***

$$\mathcal{P}_x(x' \le x) = \int_{-\infty}^{x} f_x(x')\mathrm{d}x'$$

kumulativ



- Sind die zwei Zufallsvariablen x und y **unabhängig**, dann gilt für die Wahrscheinlichkeitsdichte: $f(x,y) = f_x(x) \cdot f_y(y)$

Kovarianz und Korrelation

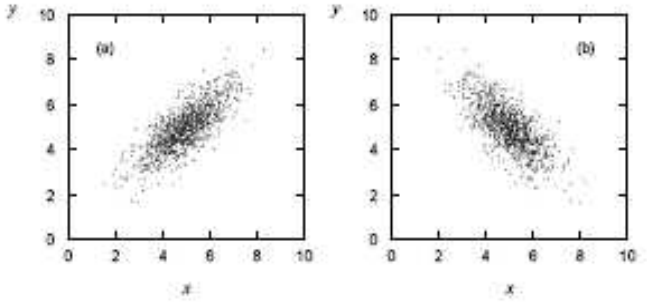
Definition Kovarianz: $\mathrm{cov}_{i,j} = E[(x_i - \mu_i)(x_j - \mu_j)] = \ldots = E[x_i x_j] - \mu_i \mu_j$

Normiere so, dass Diagonalelemente = 1

⇒ **Korrelationskoeffizenten :** $\rho_{xy} = \dfrac{\mathrm{cov}(x,y)}{\sigma_x \sigma_y}\,, \quad -1 \le \rho_{xy} \le 1$

Streudiagramme von zwei korrelierten Zufallsvariablen

$\rho = 0.75$



$\rho = -0.75$

$\rho = 0.95$

$\rho = 0.25$

Korrelationskoeffizienten ρ_{ij} sind anschaulicher als Kovarianzen !

x, y
- unkorreliert ≠ unabhängig
- korreliert ≠ kausal verknüpft
- Korrelation kann zufällig sein,
  * x ⇒ y möglich
  * y ⇒ x möglich
  * z ⇒ (x, y) möglich

Kovarianzmatrix von korrelierten Messungen

Mehrere (n) Messwerte m_i , jede Messung hat

- eine individuelle, zufällige Unsicherheit z_i
- sowie eine gemeinsame Unsicherheit z_g

$\rightarrow \quad m_i = w + z_i + z_g$

ein gemeinsamer wahrer Wert und n+1 Zufallszahlen

(eine allen Messungen gemeinsame sowie n individuelle)

Für n Messwerte m_i erhält man die Kovarianz-Matrix, indem man die Varianz der gemeinsamen Unsicherheit z_g, $(\sigma_g)^2$, in die Nebendiagonale setzt. Die Diagonalelemente sind die Varianzen der Gesamtunsicherheiten, $(\sigma_i)^2 + (\sigma_g)^2$.

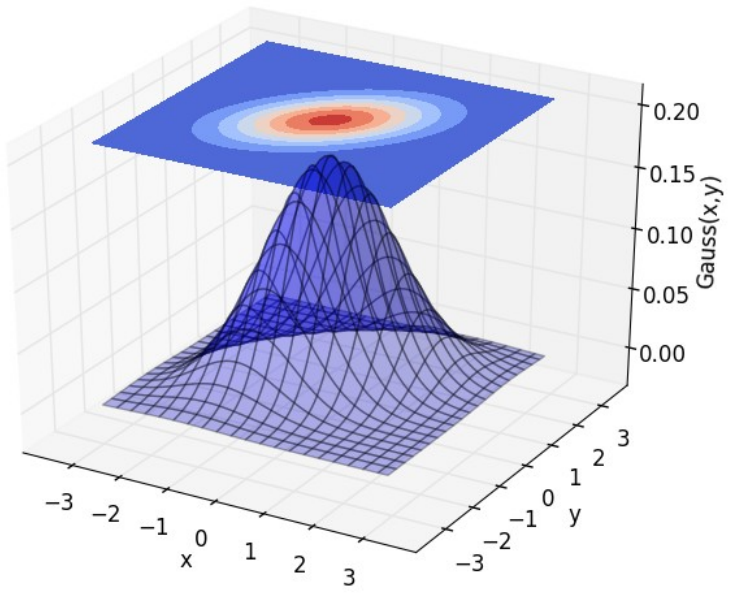
Die vollständige Kovarianzmatrix sieht so aus:

$$V = \begin{pmatrix} {\sigma_1}^2 + {\sigma_g}^2 & {\sigma_g}^2 & \dots & {\sigma_g}^2 \\ {\sigma_g}^2 & {\sigma_2}^2 + {\sigma_g}^2 & \dots & {\sigma_g}^2 \\ & \dots & & \\ {\sigma_g}^2 & {\sigma_g}^2 & \dots & {\sigma_n}^2 + {\sigma_g}^2 \end{pmatrix}$$

Multidimensionale Gaußverteilung

$$p(\vec{x}|\vec{\mu}, V) = \frac{1}{(2\pi)^{\frac{n}{2}}|V|^{\frac{1}{2}}} \exp\left[-\frac{1}{2}(\vec{x}-\vec{\mu})^T V^{-1}(\vec{x}-\vec{\mu})\right]$$

2d Gauß $\mu_i=0,\ \sigma_i=1,\ \rho=0.5$



Script: `plot3dGauss.py`

$$V = \begin{pmatrix} \sigma_1^2 & \sigma_1\sigma_2\rho_{12} & \cdot & \cdot & \cdot & \sigma_1\sigma_n\rho_{1n} \\ \sigma_2\sigma_1\rho_{21} & \sigma_2^2 & \cdot & \cdot & \cdot & \sigma_2\sigma_n\rho_{2n} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \sigma_n\sigma_1\rho_{n1} & \cdot & \cdot & \cdot & \sigma_n\sigma_{n-1}\rho_{n-1} & \sigma_n^2 \end{pmatrix}$$

allg. Kovarianzmatrix
mit Korrelationskoeffizienten ρ_{ij}

2-dimensional

$$g(x_1, x_2 \,|\, \mu_1, \mu_2, \sigma_1, \sigma_2, \rho) = \frac{1}{2\pi\sigma_1\sigma_2\sqrt{1-\rho^2}}$$

$$\cdot \exp\left(-\frac{1}{2(1-\rho^2)}\left[u_1^2 + u_2^2 - 2\rho u_1 u_2)\right]\right)$$

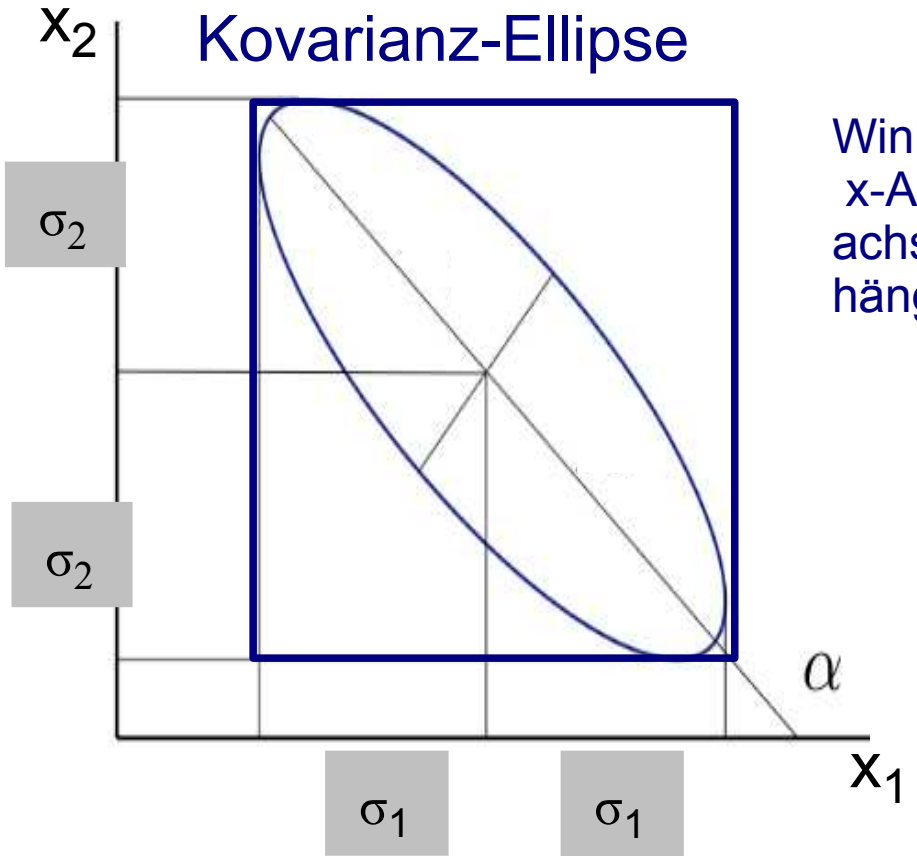
$$\text{mit } u_i = \frac{x_i - \mu_i}{\sigma_i}$$

Kovarianz-Ellipse

Kontur konstanter Wahrscheinlichkeitsdichte („Höhenlinie“) bei 2d-Gaußverteilung

$$\left(\frac{x_1-\mu_1}{\sigma_1}\right)^2+\left(\frac{x_2-\mu_2}{\sigma_2}\right)^2-2\rho_{12}\left(\frac{x_1-\mu_1}{\sigma_1}\right)\left(\frac{x_1-\mu_2}{\sigma_2}\right)=1-\rho_{12}^2$$

ist eine **Ellipsengleichung**



Winkel zwischen x-Achse und Haupt-achse der Ellipse hängt von ρ_{12} ab:

$$\tan 2\alpha=\frac{2\rho_{12}\sigma_1\sigma_2}{\sigma_1^2-\sigma_2^2}$$

$\alpha=0°$ für $\rho_{12}=0$

Fläche innerhalb der 1-σ Ellipse entspricht ~39% Wahrscheinlichkeit

gebräuchliche Darstellung bei Problemen mit vielen Variablen:
Kovarianz-Ellipsen von Variablen-Paaren

Monte Carlo-Methode zur Simulation

Die **Monte-Carlo-Methode** (auch „MC-Simulation“)

- abgeleitet vom Namen der durch ihr Spielkasino berühmten Stadt Monte Carlo -

ist eine auf **(Pseudo-)Zufallszahlen basierende numerische Methode** zur

- Integration in hochdimensionalen Räumen
- Bestimmung der Eigenschaften von Verteilungen von Zufallszahlen
  (z.B. von Messgrößen in Experimenten)
- Nachbildung von komplexen Prozessen
  (z.B. in Thermodynamik, Wirtschaft oder von experimentellen Apparaturen, u.v.a.)

Grundsätzliche Vorgehensweise:

I. Erzeuge Folge von Zufallszahlen $r_1, r_2, \ldots, r_m$, gleichverteilt in]0,1].

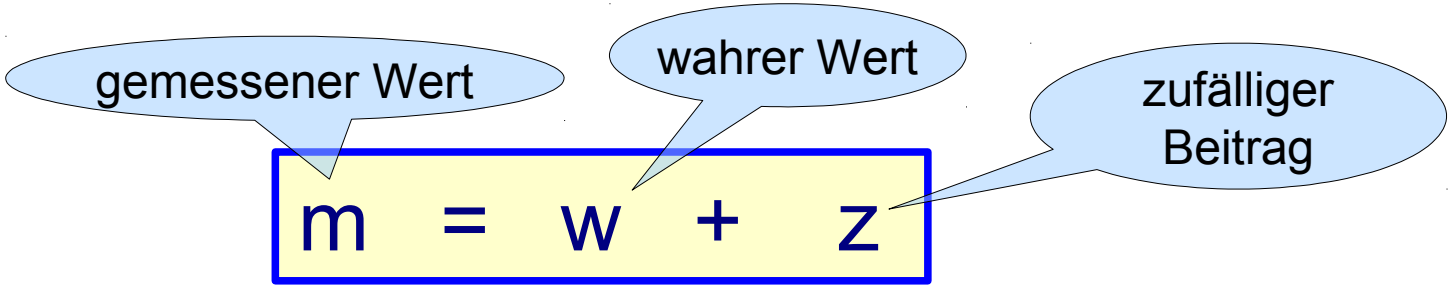
II. Transformiere diese zur Erzeugung einer anderen Sequenz $x_1, x_2, \ldots, x_n$, die einer Verteilungsdichte $f(x)$ folgen (Anm.: x_i können auch Vektoren sein !)

III. Aus den x_i werden dann Eigenschaften von $f(x)$ bestimmt

Vorteil: *einfache Operationen mit den x_i ersetzen sehr viel komplexere Operationen auf den Verteilungsdichten*

Simulierte Daten

Zufallskomponente z bei der Gewinnung empirischer Daten ebenfalls mit Zufallszahlen modellierbar:



Funktion (=Modell) des wahren Wertes, ***y = f(x),*** und Modellierung der Messwerte mit MC:

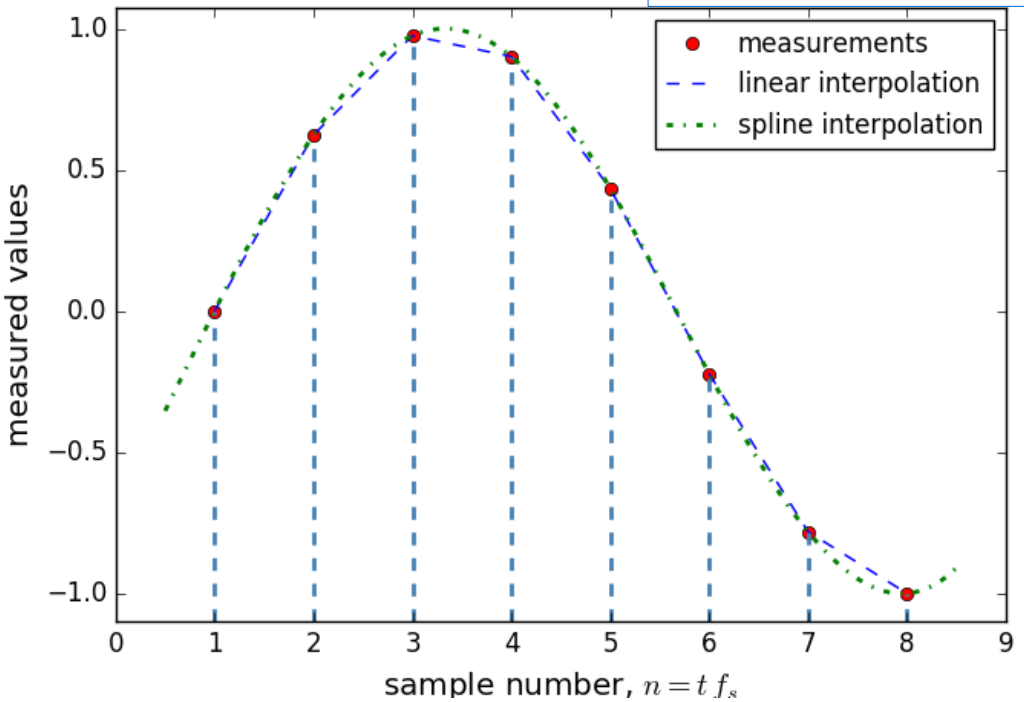
$$m_y = f(w_x + z_x) + z_y$$

z_x und z_y entsprechen den Unsicherheiten der Messgrößen x und y

Reale Daten: Digitale Abtastung („sampling“)

Digitale Messgeräte tasten Signale üblicherweise regelmäßig zu festen Zeiten t_i mit der Abtast- oder Sampling-Rate f_s ab.

Script `sampling.py`



Nummer der Messung

$$n = t_i \cdot f_s$$

Datenexport
in Felder mit
- Zeitpunkten t_i `( t[0, …,N-1] )`
- Messwerten v_i `( v[0, …,N-1])`

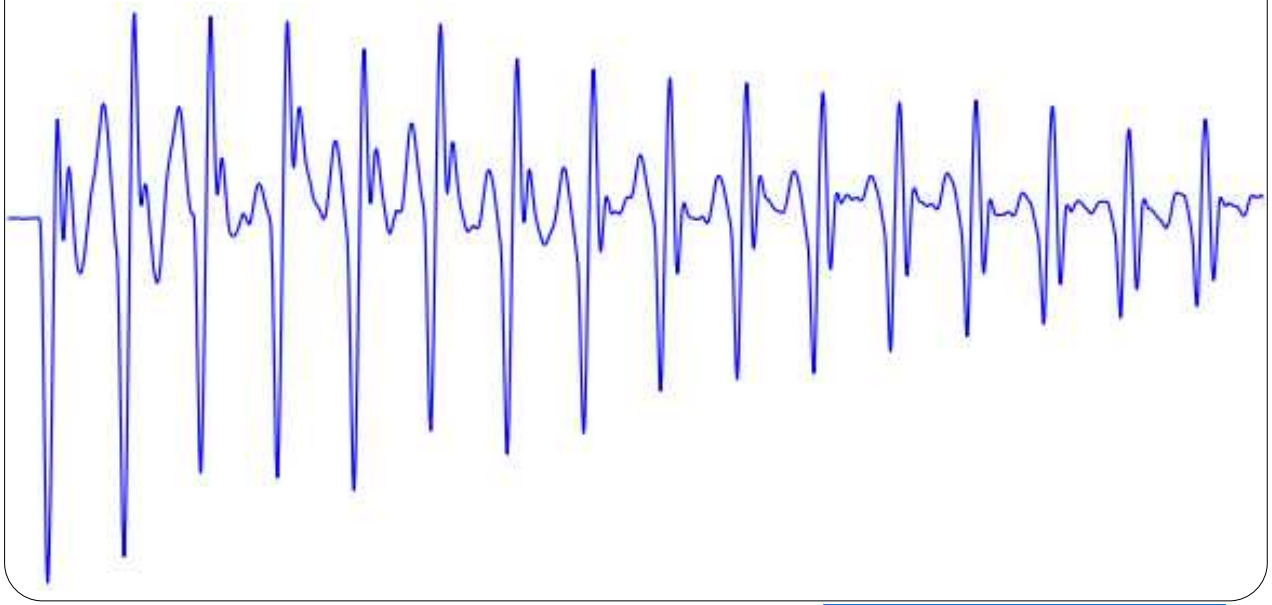
Grafische Darstellung als Markierungen, evtl.
– verbunden durch gerade Linien „lineare Interpolation“
– verbunden durch Kurven, z.B. „Spline-Interpolation“
(kubische Splines:
stetig differenzierbar aneinander anschließende Polynomstücke 3. Grades)

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)`



Datei
Welleenform.csv

Script:
test_readPicoscope.py

(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)

-0.349,-0.000458
-0.348,-0.000458
-0.347,-0.000458
    . . .
```

} Kopfzeilen mit sog. „**Meta-Daten**“

} die eigentlichen **Daten** als Dezimal-zahlen, 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“ Datenformate, z.B.
xml = „**e**xtensible **m**arkup **l**anguage“ oder
json = „**j**ava **s**cript **o**bject **n**otation“ (≙ python dictionary);
gut durch pyhton-Module unterstützt !

Rein „binäre“ Datenformate (also sehr kompakte Darstellungen in Maschinen-abhängigem 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 2D-numpy-array einlesen
data = np.loadtxt(f,
        delimiter=',', unpack=True)
print "-> Anzahl Spalten",
        data.shape[0]
print "-> Datenzeilen",
        data.shape[1]

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

s. auch
- PhyPraKit.readCSV()
- PhyPraKit.readtxt()
- PhyPraKit.readColumnData()
- PhyPraKit.labxParser()

u. a.

Bearbeitung/Analyse abgetasteter Daten

- Glätten durch gleitenden Mittelwert oder andere Filter
- Datenreduktion
  - durch Mittelwertbildung über *n* Datenpunkte
  - Selektion von Untermengen, d. h.
    nur jeden *n*-ten Punkt verwenden
- Symmetrisieren der Daten, d.h. Offset-Korrektur:
  $y_i \rightarrow y_i$ – **Mittelwert**(y_i)
- Extraktion bestimmter Eigenschaften:
  - Minima bzw. Maxima
  - Sprünge bzw. Stellen maximaler Steigung
    Faltung mit geeigneter Filterfunktion
- Frequenzbestimmung bei periodischen Signalen
  Autokorrelation
- Interpolation
  kubische Splines
- Frequenzanalyse:
  Fourier-Transformation

s. Funktionen in PhyPraKit
und die Beispiele
- `test_convolutionPeakFinder()`
- `test_AutoCorrelation.py`
- `test_Fourier.py`
- `UnivariteSpline.py`

u. a.

Frequenz eine periodischen Wellenform

Autokorrelation

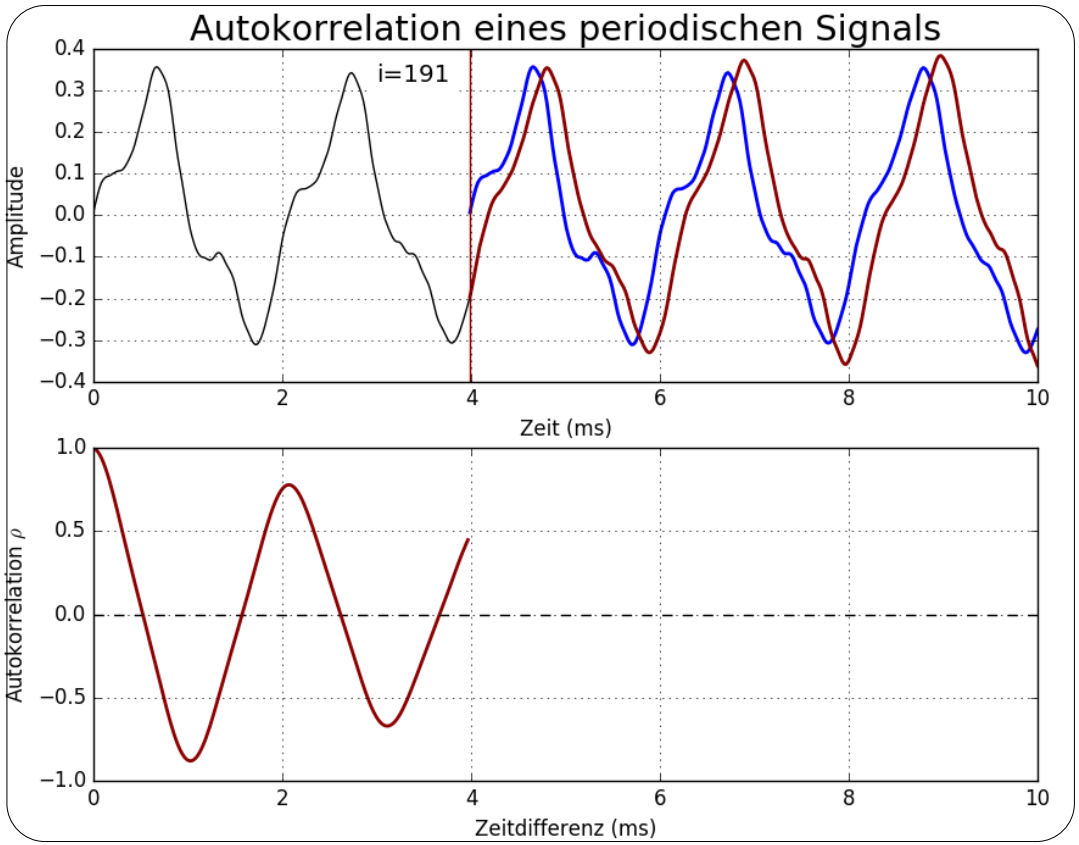
Faltung mit **verschobener Version** von sich selbst:

$$\rho_0 = \sum_{k=0}^{l-1} a_i\, a_i\ ,\ \rho_i = \frac{1}{\rho_0} \sum_{k=0}^{l-i-1} a_k\, a_{i+k},\ i=1,\ \ldots,\ \text{n-1}$$

Wegen der Selbstähnlichkeit der jeweiligen Perioden nimmt ρ Maxima an, wenn die Verschiebung einer Periodenlänge entspricht

```
# einfache Implementierung
  ...
l=len(a)
rho=np.zeros(l)
for i in range(1, l):
  rho[i]=np.inner(a[:-i],a[i:])
rho[0] = np.inner(a, a)
rho = rho/rho[0]
  ...
```

`np.inner(a,b)` *kennen wir in der Mathematik als das Skalarprodukt von a und b*

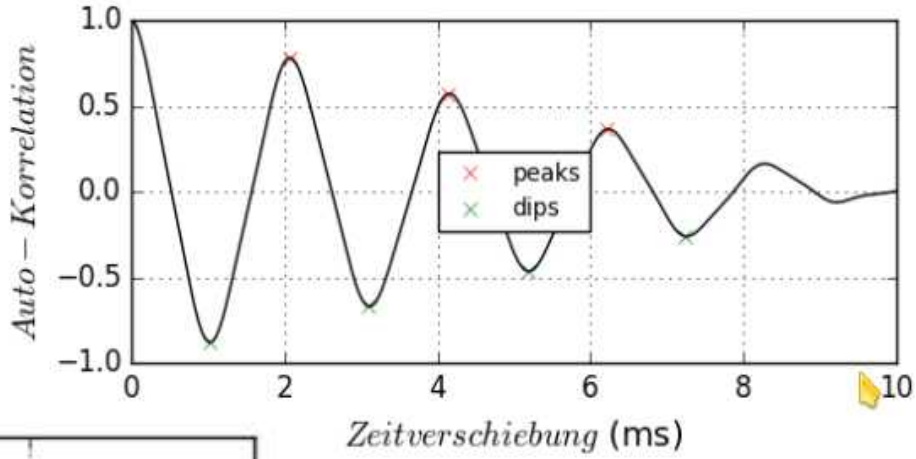


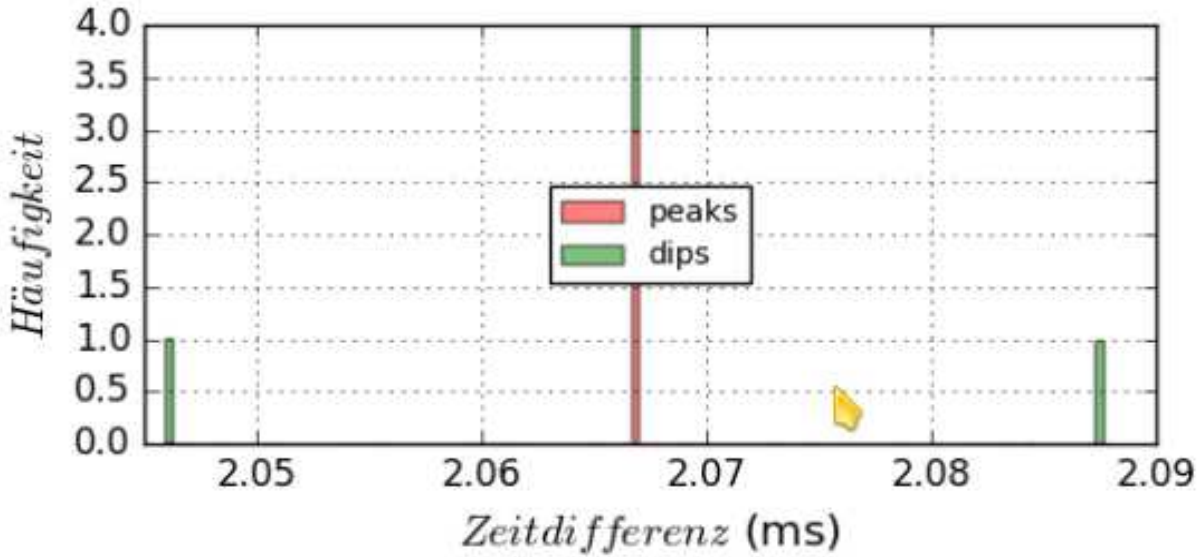
animate_autoCorrelation.py

Frequenz eine periodischen Wellenform (2)

Bestimmung der Maxima und Minima der Autokorrelation und Differenzbildung liefert sehr genaue Bestimmung der

Periodendauer





Beispiel-Skript:
`test_AutoCorrelation.py`

Zeitdifferenz T aus Maxima: (2.0668 +/- 0 [0,0060]*)) ms
bzw. Minima: (2.0668 +/- 0.0049) ms der Autokorrelation

Anm.: Unsicherheiten aus Unsicherheit des Mittelwerts („ $\sigma/\sqrt{n}$“);
*) Die Unsicherheit von 0 auf T aus Maxima bedeutet, dass Abweichungen kleiner sind als die Sampling-Zeit t_s = 0.0208 ms; als Unsicherheit wurde daher die Standardabweichung einer Rechteckverteilung mit Breite 0.0208 ms angenommen

Parameterschätzung mit der Methode der kleinsten Quadrate

(„χ^2-Methode")

Minimiere

„Summe der Residuen-Quadrate" S

bzgl. der k Parameter $\mathbf{p}$

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

σ_i^2 sind Varianzen der N Messungen y_i

Falls die Fehler korreliert sind, ersetze

$1/\sigma_i^2 \rightarrow V^{-1}$ (Inverse der Kovarianzmatrix)

$S_{\min} = S(\hat{\mathbf{p}})$ folgt

bei gaußverteilten Fehlern σ_i

einer **χ^2-Verteilung** mit

$n_f = N - k$ Freiheitsgraden

Erwartungswert: $\langle S_{\min} \rangle = n_f$

bzw. $\langle S_{\min} / n_f \rangle = 1$

χ^2-Wahrscheinlichkeit:

$$\chi^2_{\text{prob}}(S_{\min}) := \int_{S_{\min}}^{\infty} \chi^2(s, n_f) ds = 1. - \int_{0}^{S_{\min}} \chi^2(s, n_f) ds$$

dient zur Quantifizierung der Qualität einer Anpassung

Aussage, mit welcher Wahrscheinlichkeit ein größerer Wert von S am Minimum als der tatsächlich beobachtete zu erwarten wäre.

Minimierung von S

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

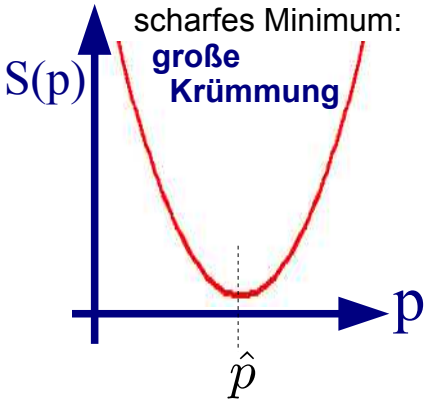
  lösbar in Spezialfällen, z. B. für lineare Probleme $f(x_i, \mathbf{p}) = \sum_{j=0}^{k} p_j\, f_j(x_i)$

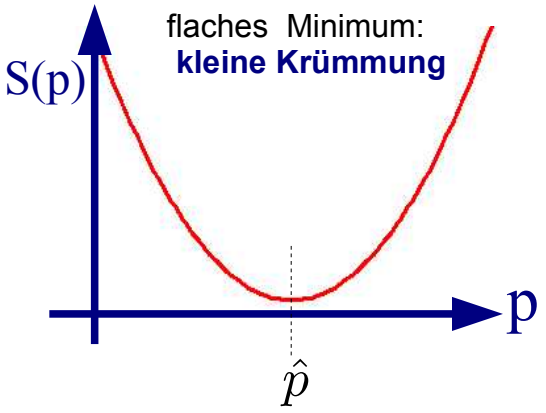
- i. a. numerisch

  „numerische Optimierung: Algorithmen zur Suche nach dem (einem?) Minimum einer Skalaren Funktion im *k*-dimensionalen Parameterraum

  z. B. **kafe:** `http://www.ekp.kit.edu/~quast/kafe/html` oder `https://github.com/dsavoiu/kafe`

Parameterunsicherheiten





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

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

Numerische Bestimmung des Minimums

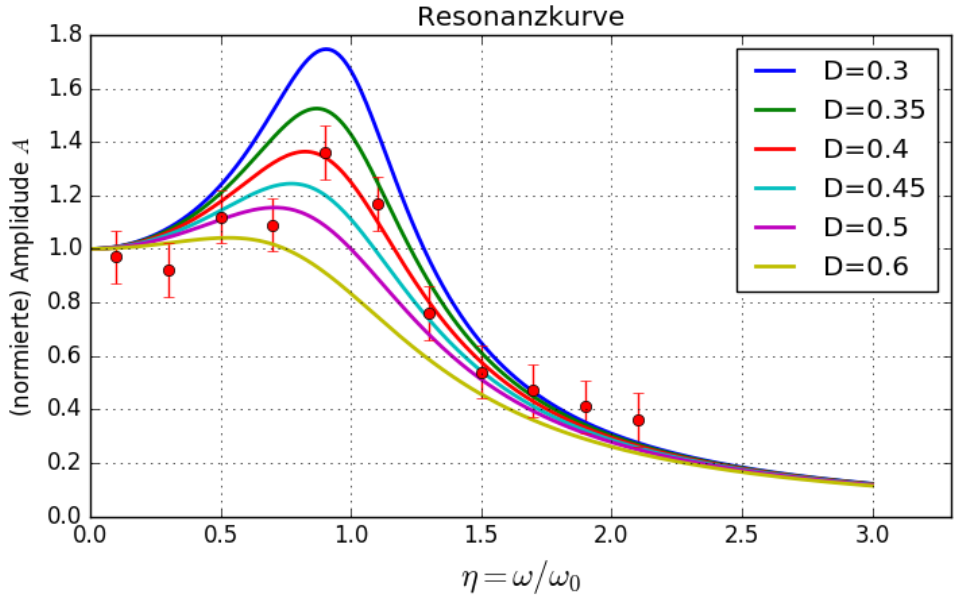
siehe Übung 4.3:

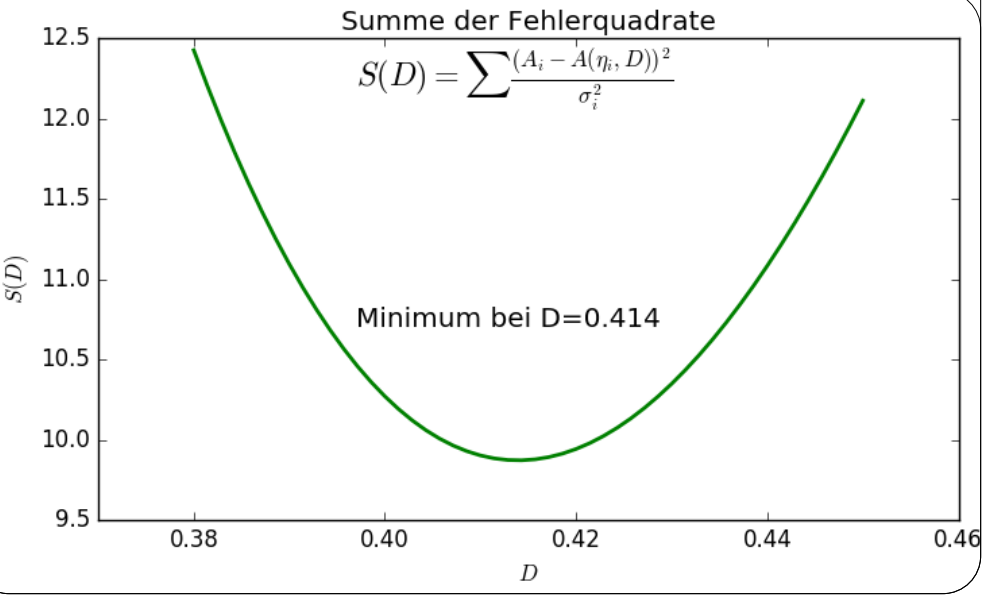
– 11 Messpunkte A_i

– Modell „Resonanzkurve“, freier Parameter Dämpfung *D*

– Abstandsmaß

$$S(D) = \sum_{i=1}^{11} \frac{(A_i - A(\eta; D))^2}{\sigma^2}$$





Die Funktion *S(D)* ist (nahezu) parabelförmig mit klarem Minimum bei *D* = 0.414

Unsicherheiten mit Toy-MC

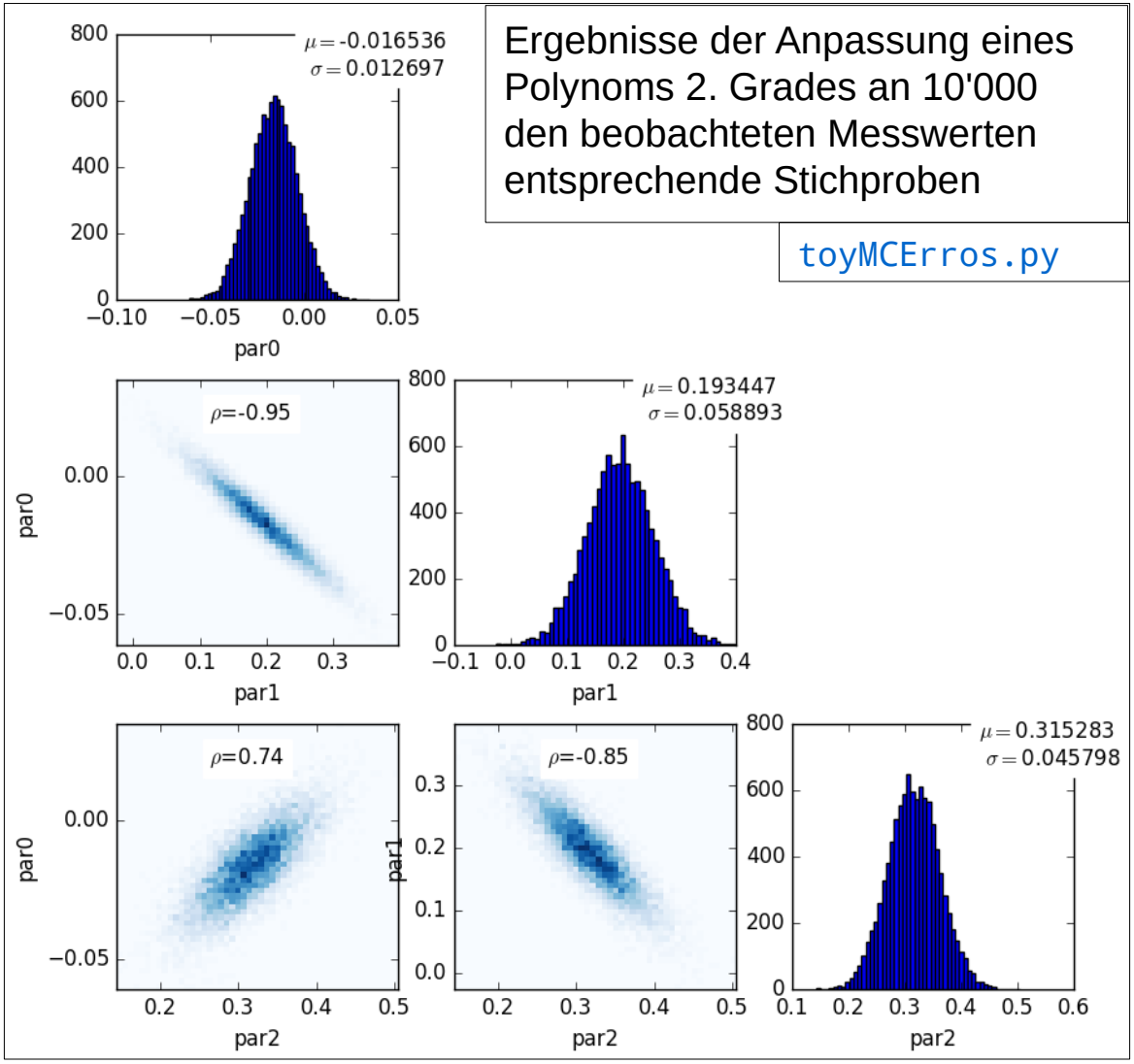
Die Unsicherheiten können ganz allgemein bestimmt werden, indem man die Anpassung für viele Stichproben wiederholt und die Verteilungen der Parameterwerte analysiert.

Auf diese Weise lassen sich
- **Parameterunsicherheiten**
- **Korrelationen der Parameter**

bestimmen.

Das Verfahren ist
aufwändig
aber
exakt

Im Zweifelsfall wird die Exaktheit statistischer Verfahren mit solchen **„Toy – Monte Carlos“** untersucht.



Ergebnisse der Anpassung eines Polynoms 2. Grades an 10'000 den beobachteten Messwerten entsprechende Stichproben

`toyMCErros.py`

Maximum Likelihood-Prinzip

Likelihood-Funktion $\mathcal{L}$:

Produkt der Werte der Wahrscheinlichkeitsdichte P_i für n unabhängige Messungen x_i :

$$\mathcal{L}(\mathbf{p}) = P(x_1|\mathbf{p}) \cdot P(x_2|\mathbf{p}) \cdot \ldots \cdot P(x_n|\mathbf{p}) = \prod_{i=1}^{n} P(x_i|\mathbf{p})$$

hängt nur noch von den Parametern ***p*** ab !

Maximum-Likelihood-Prinzip:
Der beste Schätzwert für den Parametervektor $\hat{\mathbf{p}}$ ist derjenige, der die Likelihood-Funktion $\mathcal{L}(\mathbf{p})$ maximiert

negativer Logarithmus der Likelihood-Funktion:

$$F(\mathbf{p}) = -\ln \mathcal{L}(\mathbf{p}) = -\sum_{i=1}^{n} \ln P(x_i|\mathbf{p})$$

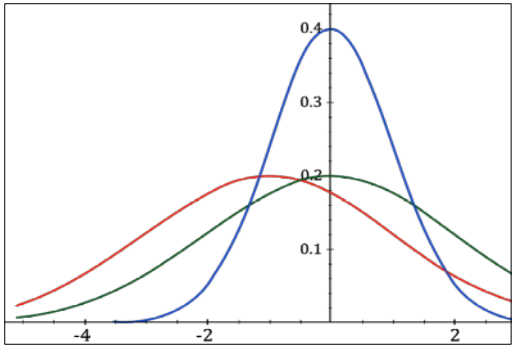
Bedingung für Optimum:

$$\hat{\mathbf{p}}: \left.\frac{dF(\mathbf{p})}{dp_j}\right|_{\mathbf{p}=\hat{\mathbf{p}}} = 0$$

Beispiel: Likelihood der Gaußverteilung

$$N(y_i, \vec{a}) = \frac{1}{\sqrt{2\pi}\sigma} \cdot \exp\left(-\frac{(y_i - f(x_i, \vec{a}))^2}{2{\sigma_i}^2}\right) \Rightarrow$$

$$-\ln\mathcal{L}(\vec{a}|\vec{y}) = \frac{1}{2}\sum_i \underbrace{\left(\frac{y_i - f(x_i, \vec{a}}{\sigma_i}\right)^2}_{= \chi^2\ !} + \underbrace{\sum_i \ln((\sqrt{2\pi}\sigma_i)}_{\text{const. bzgl. } \boldsymbol{a}}$$



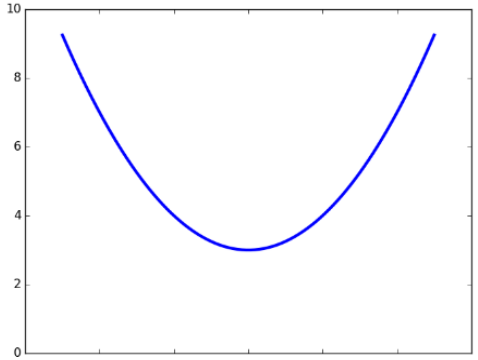
Für **Gauß-förmig** um $f(x_i, \boldsymbol{a})$ verteilte Messungen y_i ist

χ^2 äquivalent zu $-2 \ln\mathcal{L}$

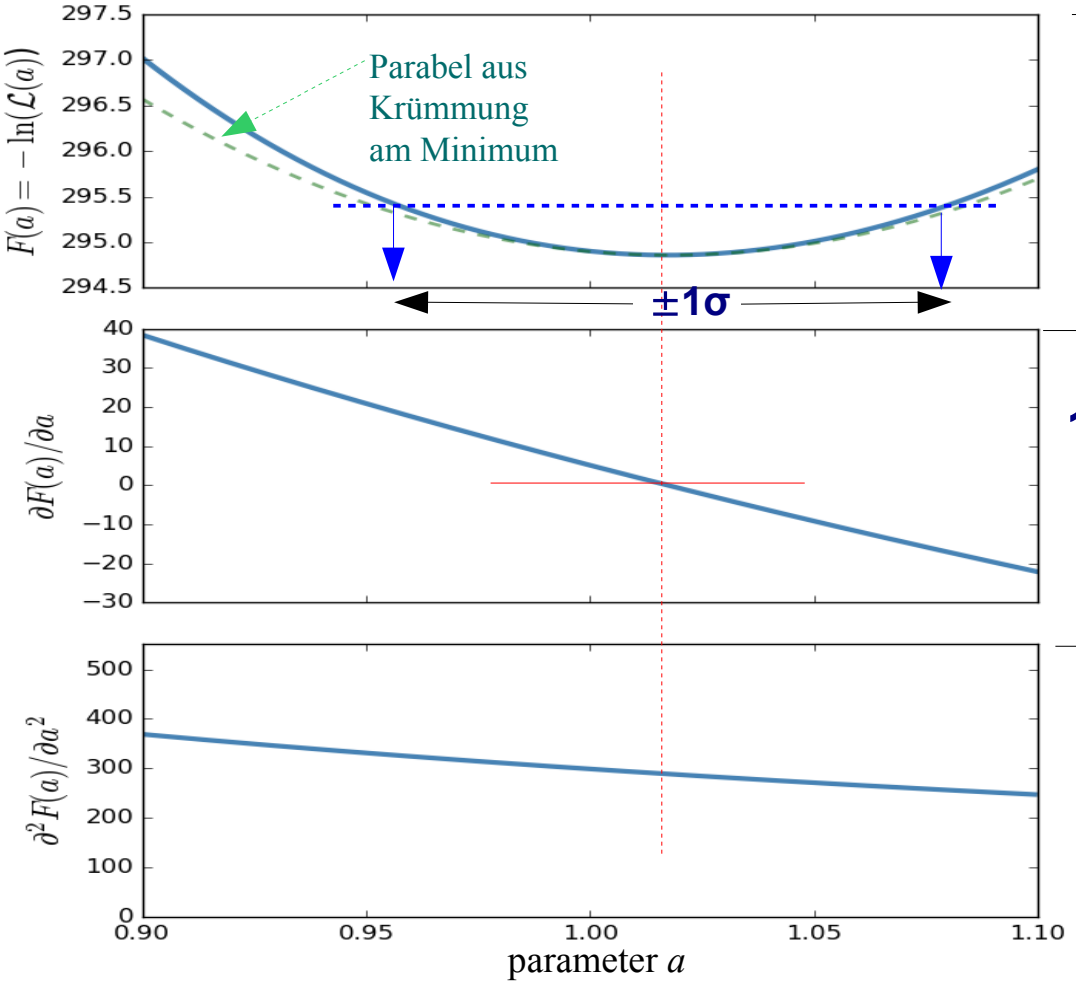
Ist $f(x_i, \boldsymbol{a})$ zusätzlich eine lineare Funktion der Parameter $\boldsymbol{a} = (a_i)$,

dann ist $-2 \ln\mathcal{L}(a_i | y_i)$ (und $\chi^2(a_i, y_i)$) eine Parabel

Näherungsweise kann $-\ln\mathcal{L}(a_i | y_i)$ in der Nähe des Minimums häufig durch eine Parabel approximiert werden !



Maximum-Likelihood: Prameterunsicherheiten



Typischer Verlauf einer negativen log-Likelihood Funktion und ihrer 1. und 2. Ableitungen

F(a) näherungsweise quadratisch um das Minimum;

±1σ - Intervall (=68%)
aus ΔF = 0.5

1. Ableitung näherungsweise linear, 0 am Minimum

2. Ableitung ~ konstant

Varianz ≈ 1 / Krümmung

$1/\sigma^2 \approx \partial^2 F \,/\, \partial a^2$

bei mehreren Parametern a_i:

$(cov^{-1})_{ij} \approx \partial^2 F \,/\, \partial a_i \, \partial a_j$

Nur für Parabel-förmigen Verlauf von $F(\boldsymbol{a})$ sind die beiden Fehlerdefinitionen äquivalent

Mathematisch exakt: die angegebenen Fehlerabschätzungen sind Untergrenzen

Übersicht: Bestimmung der Parameter-Unsicherheiten

Lineare Probleme mit gaußförmigen Unsicherheiten:

- analytische Lösung möglich (χ^2-Minimierung)
- Position des Minimums gegeben durch Linearkombination der Messwerte: $\hat{\mathbf{a}} = (\mathbf{A}^\mathbf{T}\mathbf{V}^{-1}\mathbf{A})^{-1}\,\mathbf{A}^\mathbf{T}\mathbf{V}^{-1}\,\mathbf{y}$
- Varianz der Parameterschätzung durch Fehlerfortpflanzung:

$$V(\hat{\mathbf{a}}) = \left(\mathbf{A}^\mathbf{T}\mathbf{V}^{-1}\mathbf{A}\right)^{-1}$$

Nicht- lineare Probleme oder andere als gauß-förmige Unsicherheiten:

- Likelihood-Analyse:

Ausnutzen der Cramer-Rao-Frechet-Grenze:

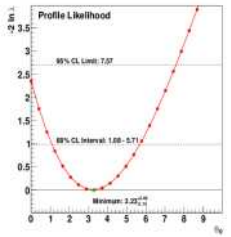
2. Ableitungen am Minimum:

$$(\hat{V}[\hat{\mathbf{a}}]^{-1})_{ij} = -\left.\frac{\partial^2 \ln L}{\partial a_i\, \partial a_j}\right|_{\mathbf{a}=\hat{\mathbf{a}}}$$

Nicht-lineare Probleme mit nicht-parabolischer Likelihood am Minimum:
(für Grenzfall hoher Statistik)

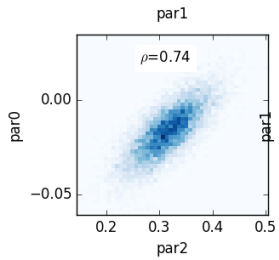
- Scan der (Profil-) Likelihood in der Nähe des Minimums

$$\ln(\mathcal{L}(\hat{a})) - \ln(\mathcal{L}(a)) = \frac{n^2}{2} \;\Rightarrow\; |a - \hat{a}| = n\sigma$$



Bei Unklarheit oder sehr kleiner Statistik:

Monte-Carlo-Studie:
Anpassung an viele der Verteilung der Daten entsprechende Stichproben, Verteilungen der Parameter auswerten.



Achtung: nur die letzten beiden Methoden liefern „Konfidenz-Intervalle“ (z. B. 1σ ≙ 68%)

kafe: Beispiele

zu **kafe** gibt es eine Anzahl von gut dokumentierten Beispielen, s.

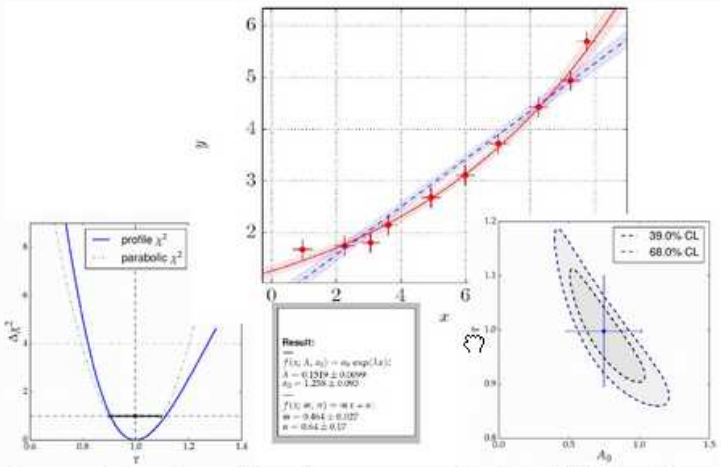
`http://www.ekp.kit.edu/~quast/kafe/html` oder `https://github.com/dsavoiu/kafe`

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*.

`kafe` Overview

The `kafe` package provides a rather general approach to fitting of a model function to two-dimensional data points with correlated uncertainties in both dimensions. The *Python* API guarantees full flexibility for data input. Helper functions for file-based input and some examples are available for own applications.



Graphical output generated with kafe.

Applications range from performing a simple average of measurements to complex situations with both correlated (systematic) and uncorrelated (statistical) uncertainties on the measurements of the x and y values described by a non-linear model function depending on a large

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.

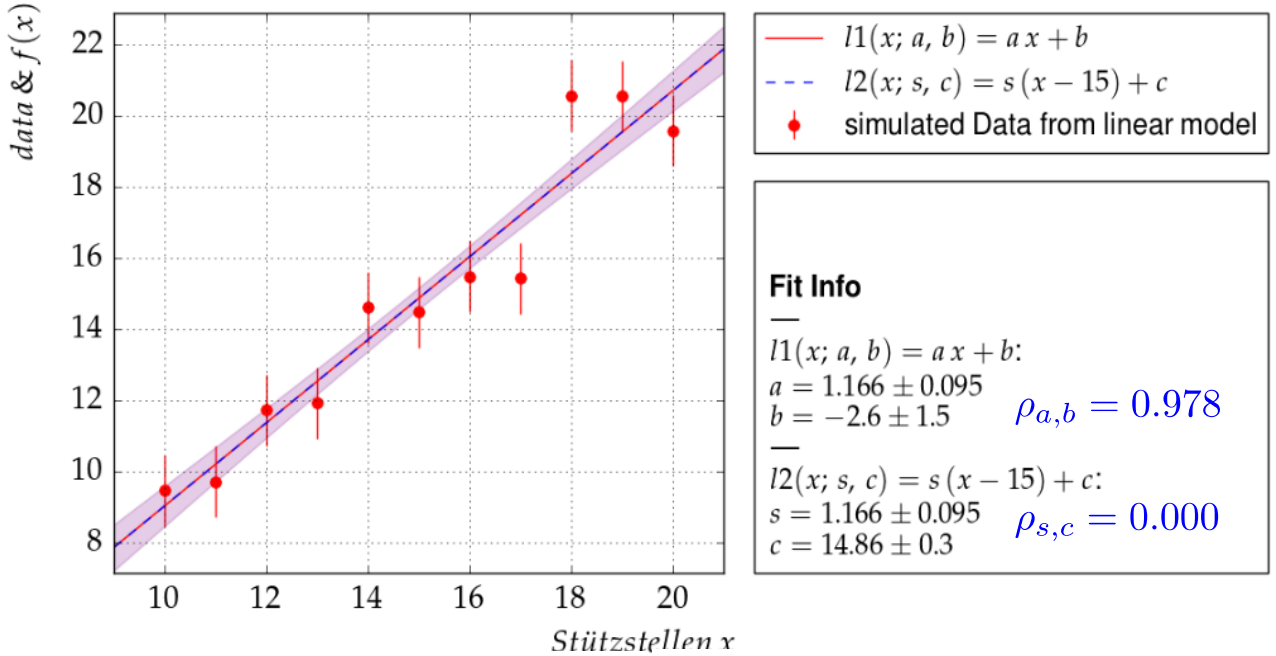
Software-Paket zur Modellanpassung

In der alltäglichen Praxis verwendet man heute Programme,
(bzw. besser **Softwarepakete**) zur Anpassung von Modellen an Parameter:

- Einlesen der Eingabedaten, evtl. Unterstützung verschiedener Formate
- Verwaltung der Unsicherheiten, evtl. für Abszisse und Ordinate,
  evtl. Kovarianzmatrizen bzw. Korrelationen
- kafe: Berücksichtigung von Einschränkungen an Modellparameter
- Bereitstellung verschiedener Fit-Funktionen, evtl. frei programmierbar
- Durchführung der Anpassung mittels numerischer Optimierung, evtl. anpassbar
- Ausgabe des Ergebnisses, als Textdatei und Grafik
- Bewertung der Qualität der Anpassung (außer χ^2 bisweilen auch andere gebräuchlich)
- evtl. Ausgabe der Korrelationen bzw. Kovarianz-Matrix der Parameter
- kafe: graphische Darstellung der Modellunsicherheit
- kafe: asymmetrische Fehler mittels Scan der χ^2 -Kurve am Minimum
- kafe: Ausgabe von Kovarianz-Ellipsen (d. h. Wahrscheinlichkeitskonturen)

Geradenanpassung: verschiedene Parametrisierungen

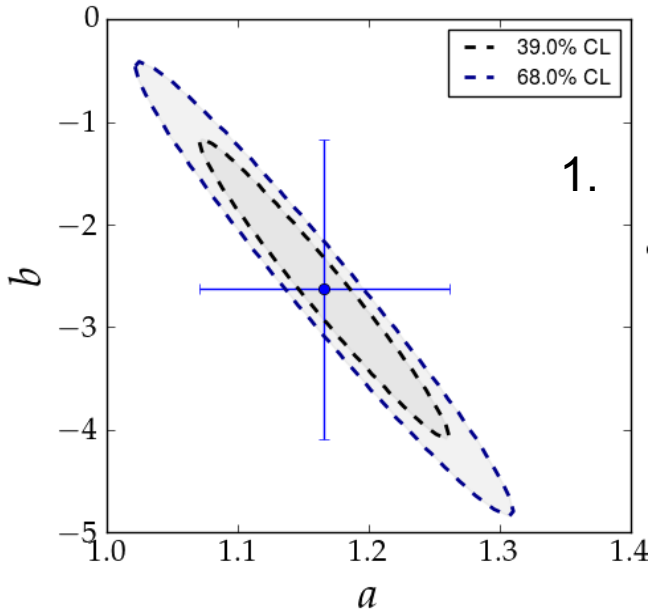
s. Übungsaufgabe 5.3

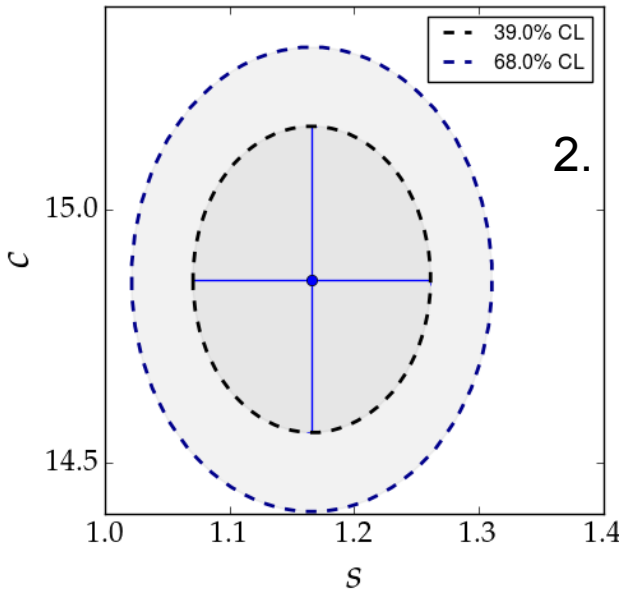


Parametrisierungen:

1. $y(x) = ax + b$

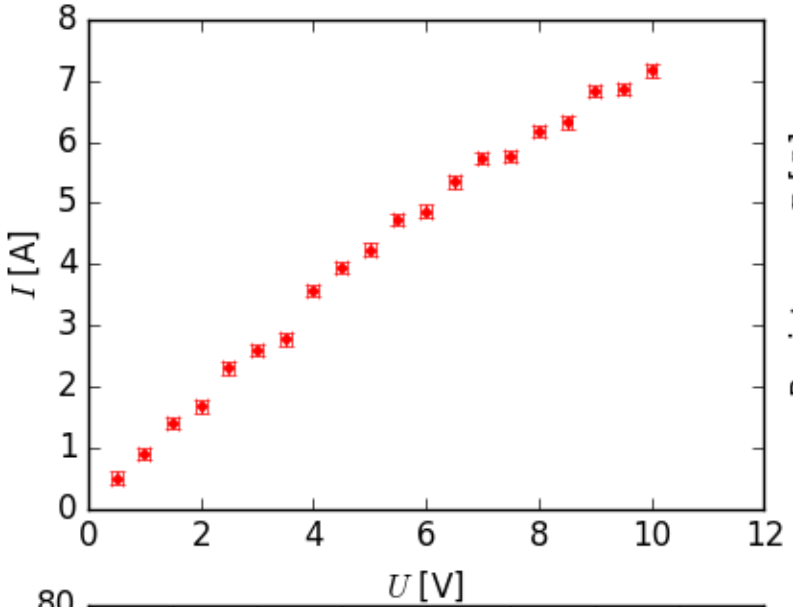
2. $y(x) = s(x - \bar{x}) + c$

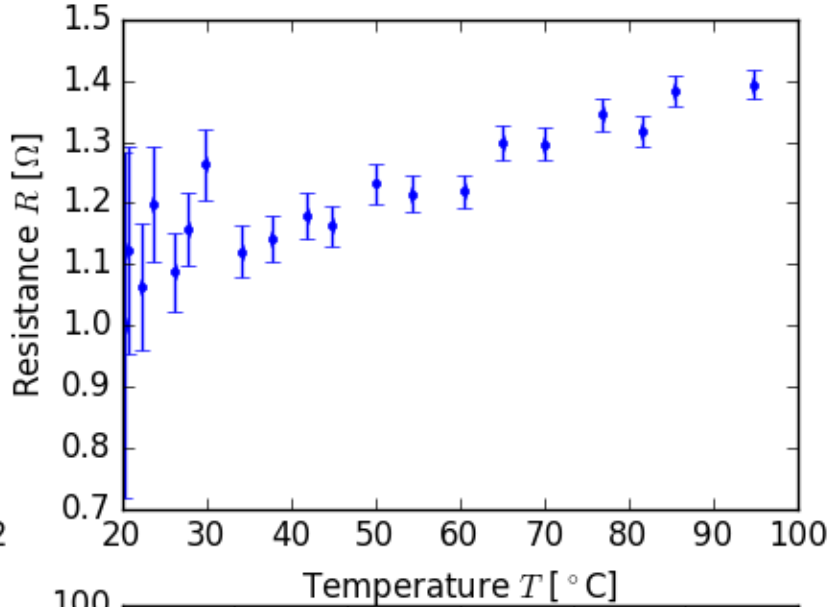


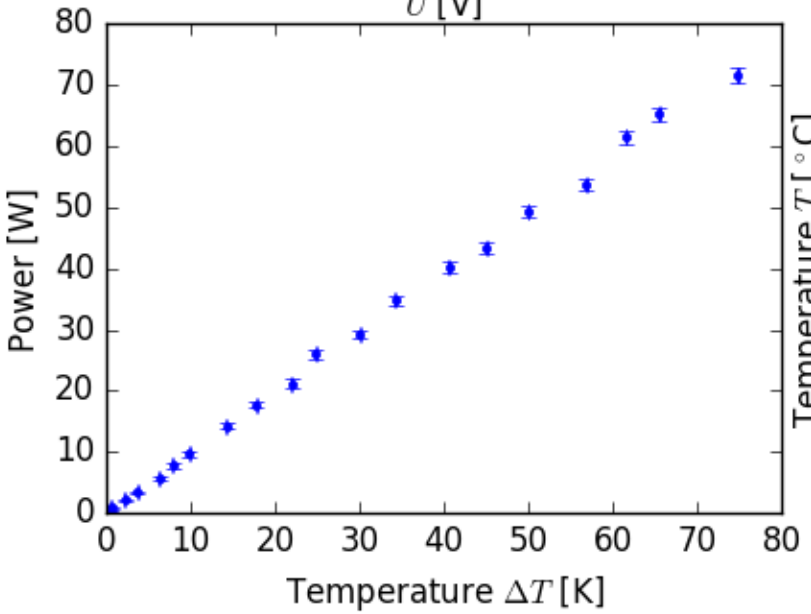


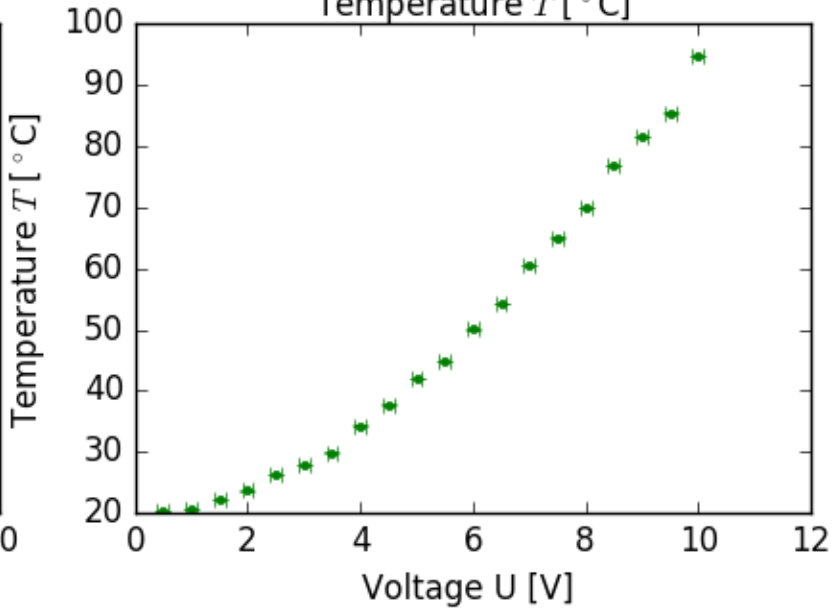
Ergebnisse für das angepasste Modell sind identisch, aber im 1. Fall sind die Parameter sehr stark korreliert.

Auswertung: Ohm'sches Gesetz





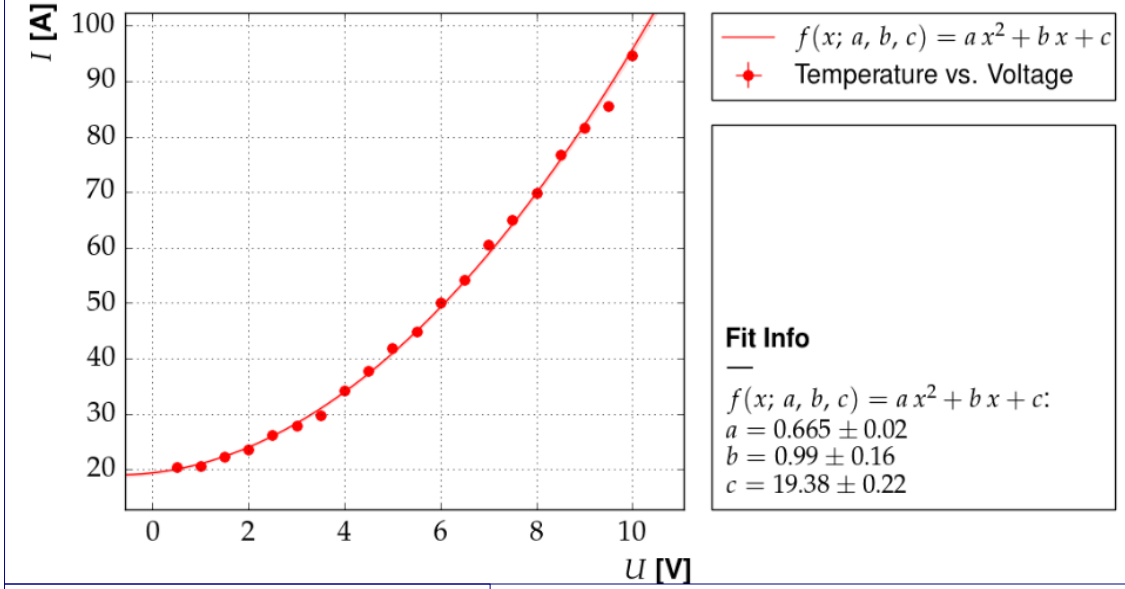




Daten erzeugt mit Script `Generate_OhmLawExperiment.py`

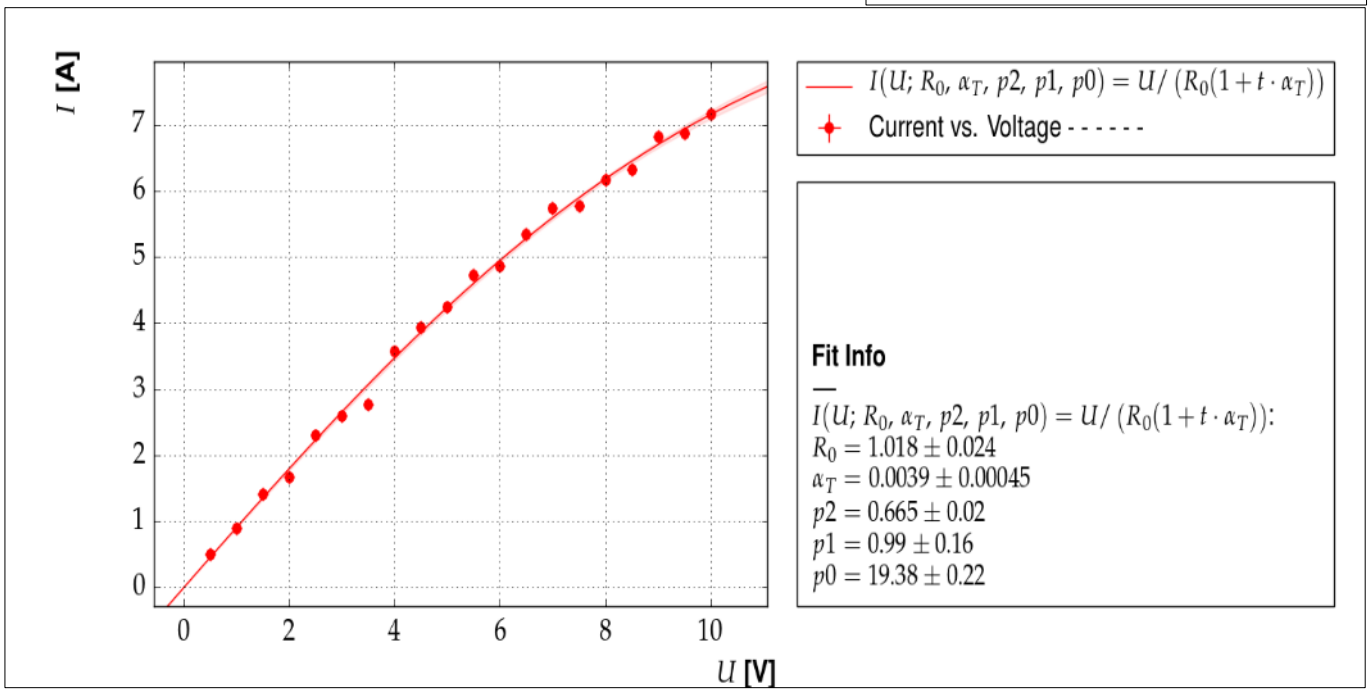
Auswertung: Ohm'sches Gesetz (2)

s. Übungsaufgabe 6



Hilfsanpassung:
Parametrisierung des Temperaturverlaufs als Funktion der Spannung

Ergebnis (Methode 2) :



Auswertung: Ohm'sches Gesetz (3)

s. Übungsaufgabe 6

Mit $p = \frac{\lambda_w(1+t\alpha_T)}{2\,\alpha_T}$ ergibt sich die Lösung:

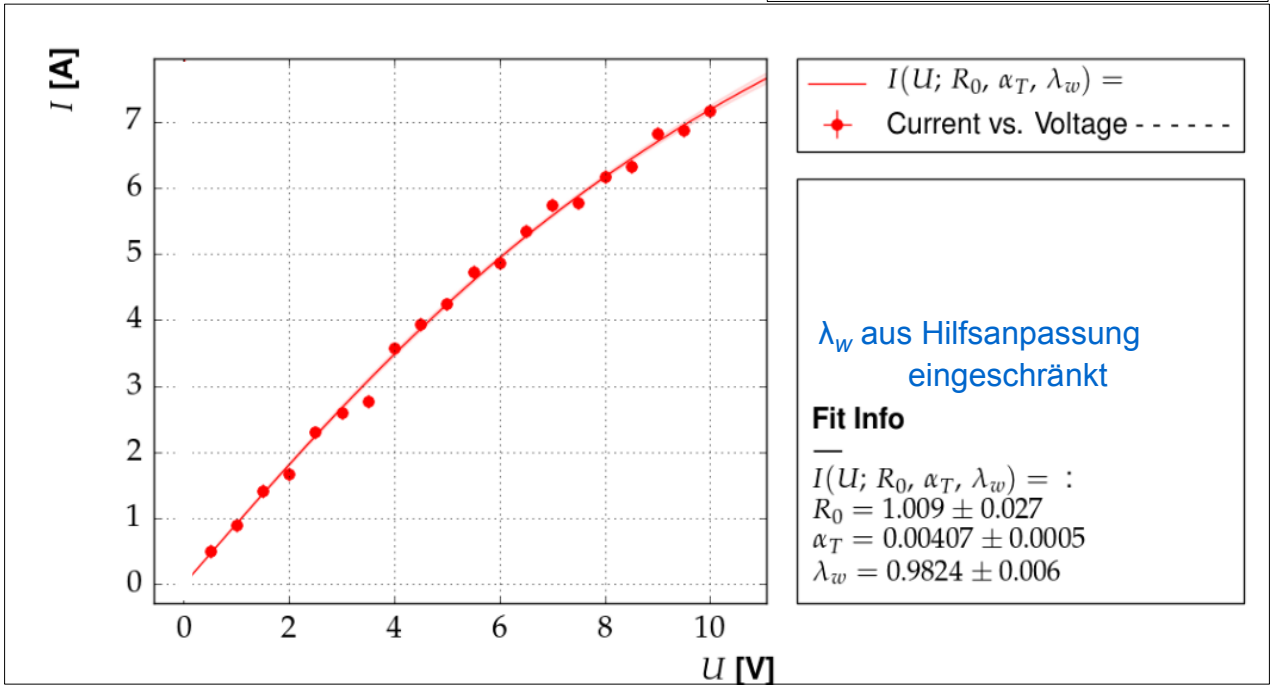
$$I(U; R_0, \lambda_w, \alpha_T) = \frac{1}{U}\left(\sqrt{p^2 + U^2 \frac{\lambda_w}{\alpha_T R_0}} - p\right)$$

Die Temperatur lässt sich mit dieser Lösung auch als Funktion der Spannung ausdrücken:

$$T(U; R_0, \lambda_w, \alpha_T) = \frac{U \cdot I}{\lambda_w} + T_{\text{env}} = \sqrt{\frac{(1+t\alpha_T)^2}{4{\alpha_T}^2} + \frac{U^2}{\lambda_w \alpha_T R_0}} - \frac{1+t\alpha_T}{2\alpha_T} + T_{\text{env}}$$

Analytisches Modell

Ergebnis (Methode 3) :



χ^2 - Methode: Do's und Don'ts

Bei Anwendung der χ^2 Methode …

nie die Messwerte transformieren

Unsicherheiten sind dann nicht mehr gaußförmig und damit die Voraussetzungen für die Methode nicht mehr gegeben

Also: $$S = \sum \frac{(f(y_i) - T(x_i))^2}{{\sigma_{f_i}}^2} \rightarrow \sum \frac{\left(y_i - f^{-1}(T(x_i))\right)^2}{{\sigma_i}^2}$$

ggf. Unsicherheiten der Therorievorhersage quadratisch addieren

Also: ${\sigma_i}^2 = {\sigma_i^{\mathrm{ex}}}^2 + {\sigma_i^{\mathrm{th}}}^2$

ggf. "Constraints" und „Fixieren" von Parametern benutzen, um externe Abhängigkeiten zu berücksichtigen

ggf. gemeinsame (systematische) Unsicherheiten mit Hilfe der Kovarianzmatrix berücksichtigen

Also: $$\sigma_i,\ \sigma_j,\ \sigma_g \rightarrow \begin{pmatrix} \sigma_i^2 + \sigma_g^2 & \sigma_g^2 \\ \sigma_g^2 & \sigma_j^2 + \sigma_g^2 \end{pmatrix}$$

χ^2 - Methode: Do's und Don'ts (2)

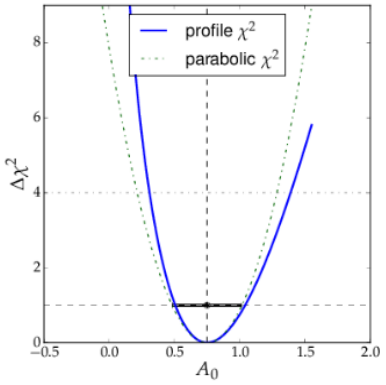
immer die χ^2-Wahrscheinlichkeit überprüfen

kafe: Variable `fit.chi2prob`

bei nicht-linearen Problemen unbedingt
den Verlauf der χ^2-Kurve am Minimum überprüfen

kafe: Methode `plot_profile()`

ggf. asymmetrischen Fehler angeben !



bei Anpassung mehrerer Parameter

immer die Kovarianzen / Korrelationen überprüfen und angeben

Kovarianzmatrix in kafe: Variable `fit.par_cov_mat`

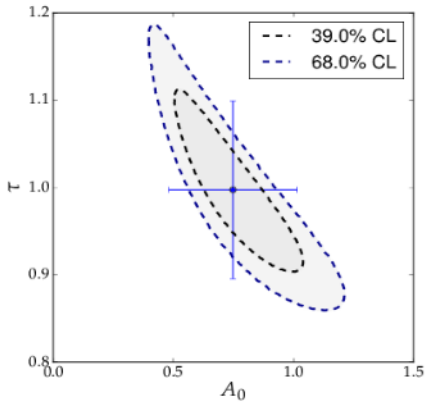
bei großen Korrelationen ggf. Parametrisierung ändern, Korrelationen angeben

(sonst keine Interpretation als Wahrscheinlichkeit möglich)

Konturen von Paaren von Parametern überprüfen

kafe: Methode `plot_contour()`

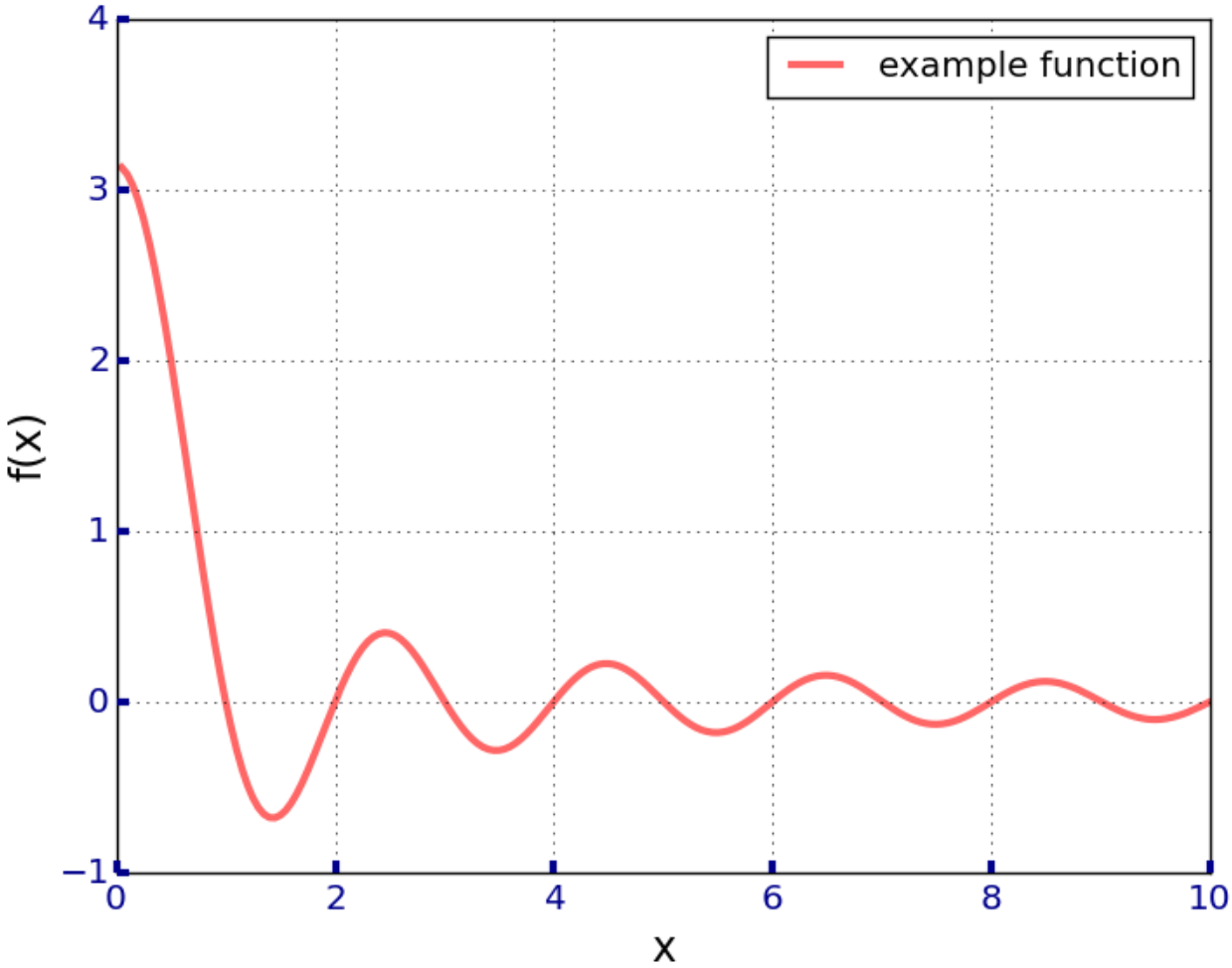
falls nicht elliptisch, Grafik veröffentlichen



Beispiel-Scripts

Darstellen von Fuktionen

Darstellung einer Funktion

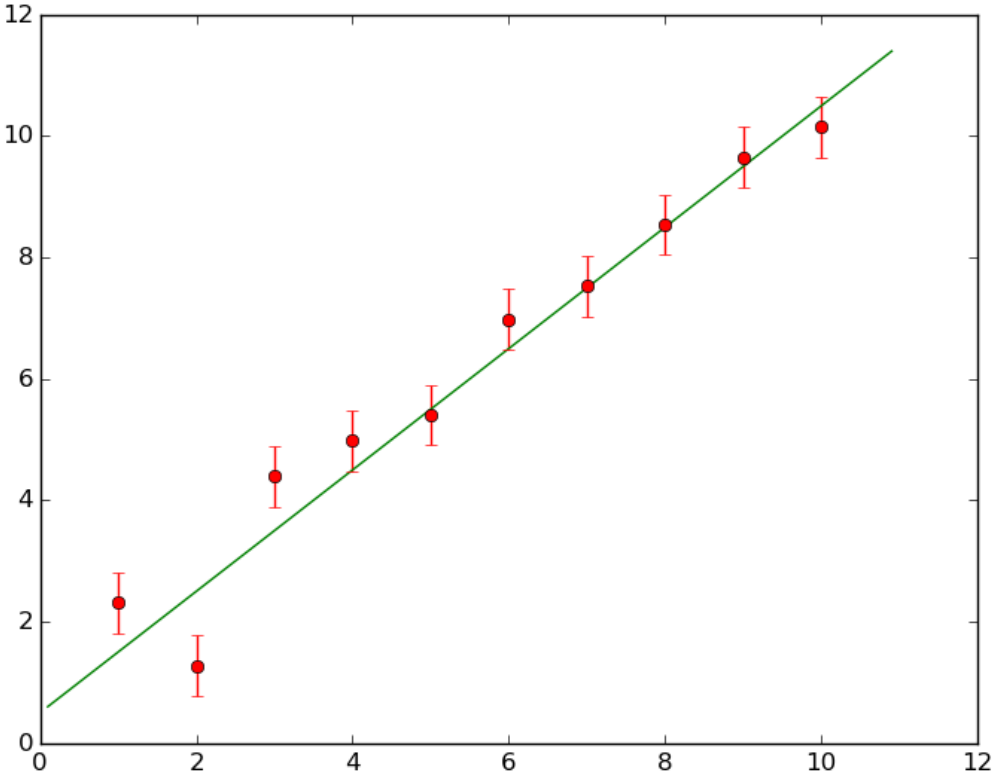


i.w. das Ergebnis dieses kurzen Programms:
siehe simplePlot.py

```
x=np.linspace(0.,10, 200)
y=np.sin(np.pi*x)/x
plt.plot(x,y,label='example function')
plt.legend()
plt.show()
```

Funktion mit Messpunkten

Datenpunkte mit Fehlerbalken
und einer Geraden

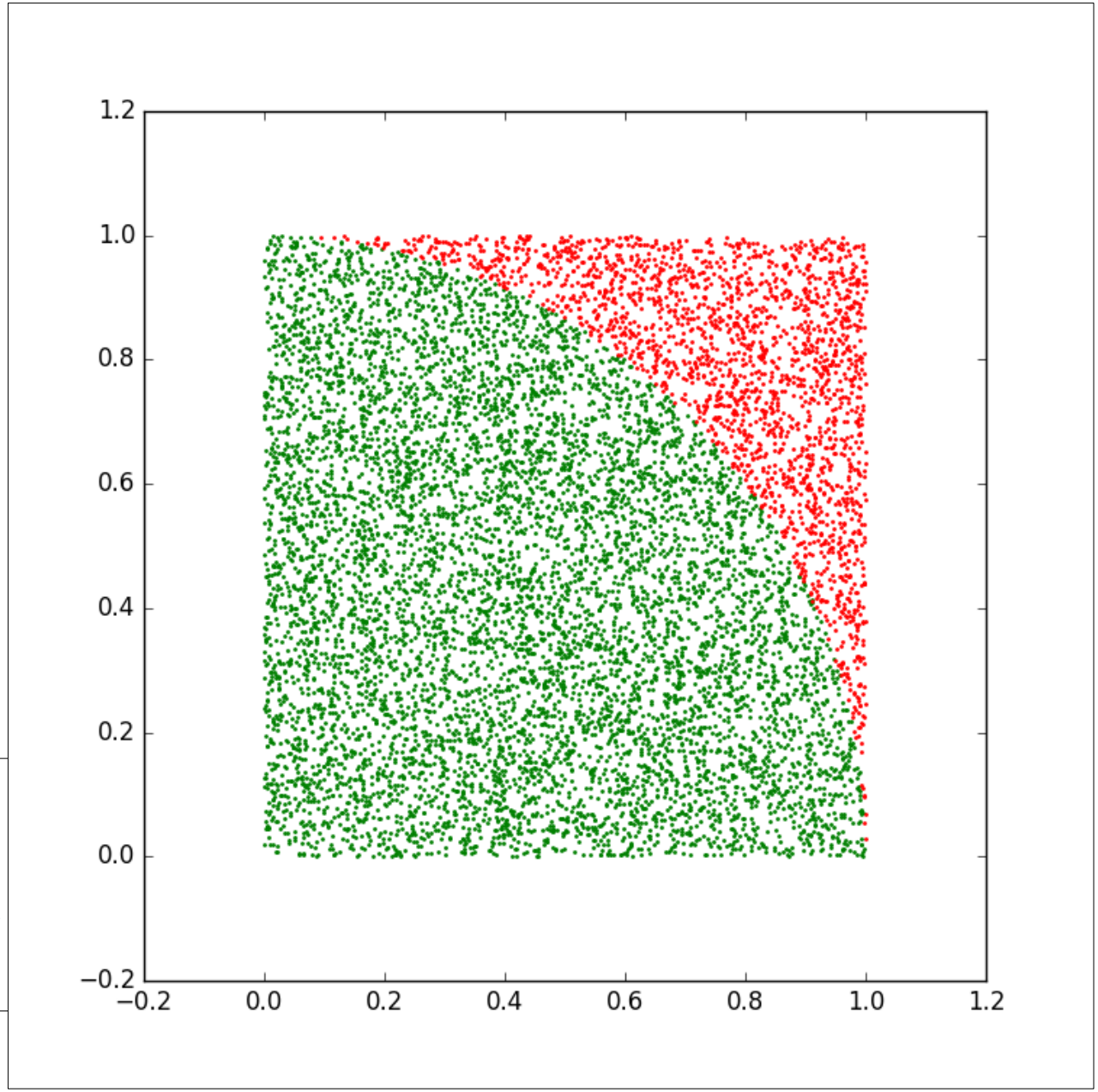



```
ym=[2.31,1.27,4.40,4.98,5.40,6.98,7.53,8.54,9.65,10.15]
xm=range(1,11)
plt.errorbar(xm,ym,yerr=0.5,fmt='ro')
x=np.linspace(0.,11.,100)
y=x+0.5
plt.plot(x,y)
plt.show()
```

simpleData.py

Streudiagramm

Bestimmung von π mit Zufallszahlen

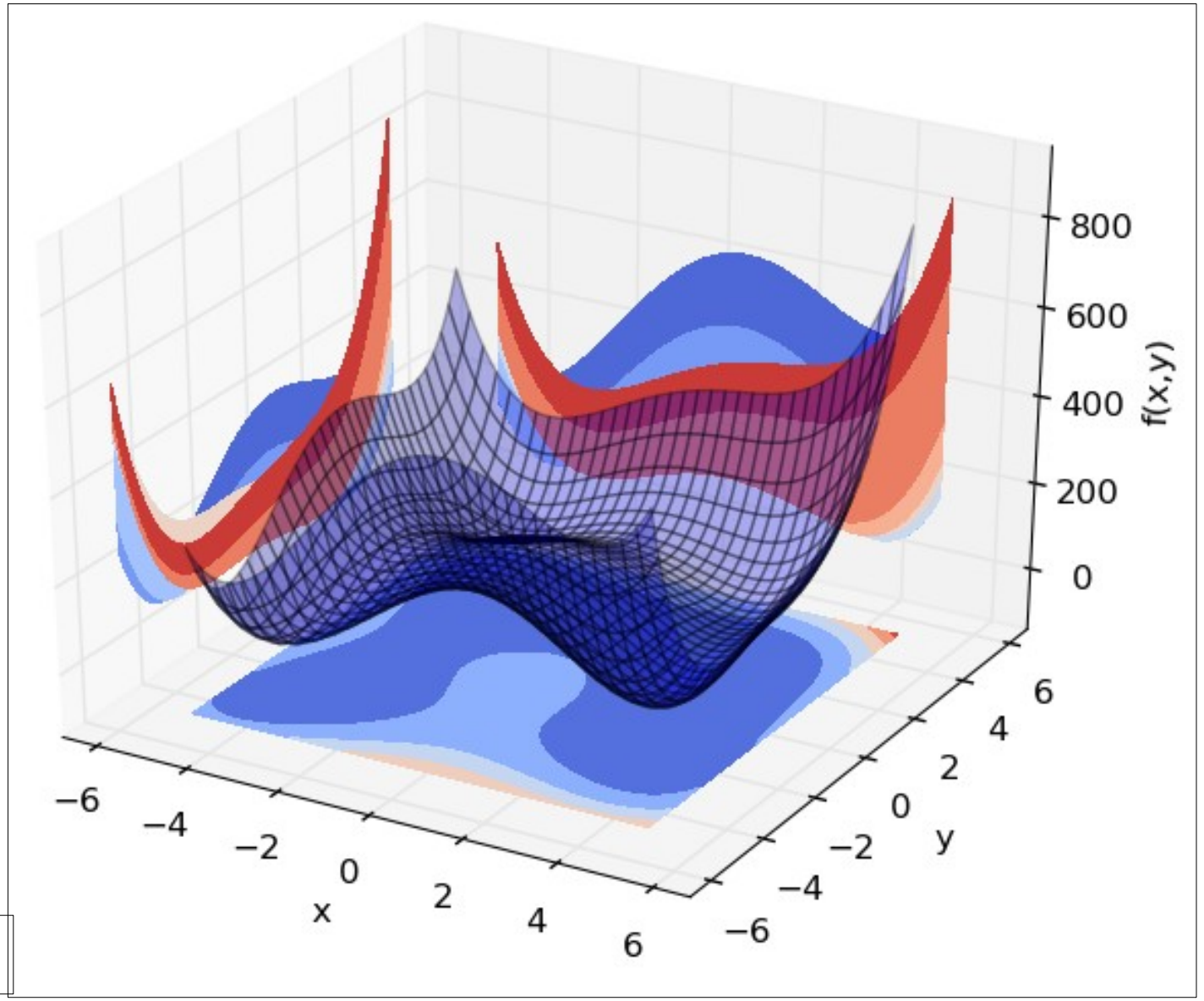


pi.py

s. auch
animate_pi.py

3d-Darstellung von Funktionen

Funktionen von
zwei Variablen in 3d

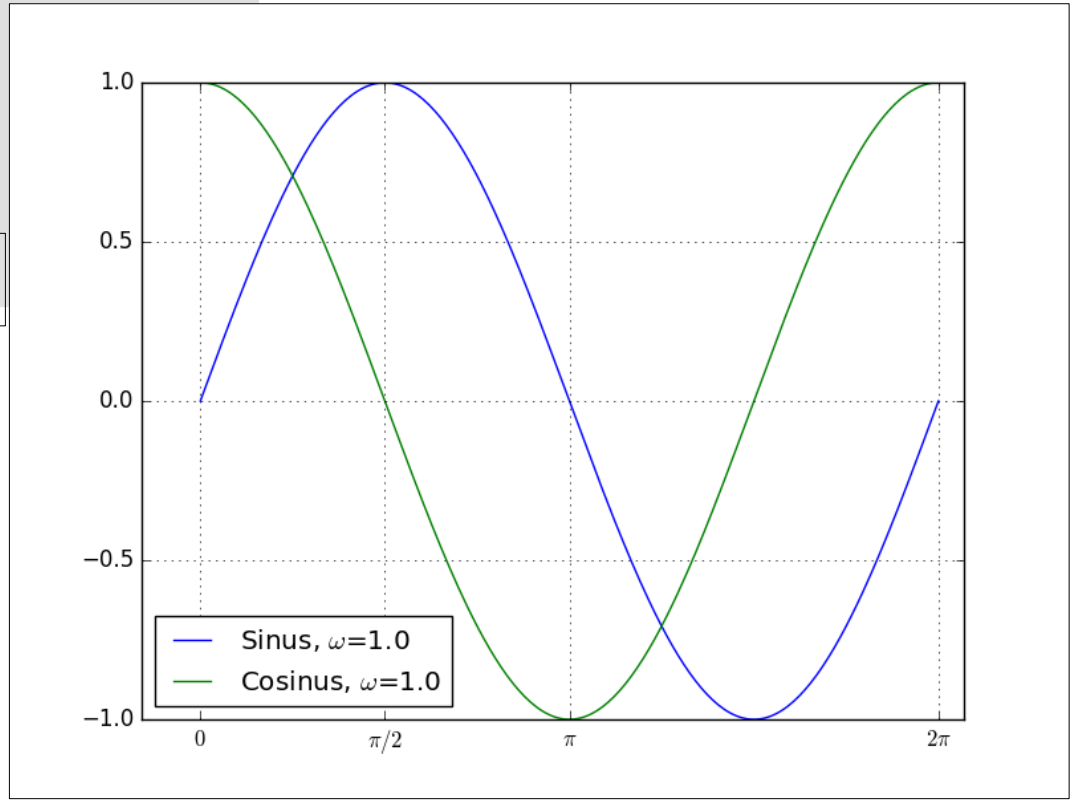


plot3dfunction.py

Allgemeine Vorlage zur Funktkionsdarstellung

```
# -*- coding: utf-8 -*-
#      diese Zeile legt die Codierung von Umlauten fest
##################################################################

# script PlotBeispiel.py
'''
   Beispiel fuer Funktionsdarstellung mit matplotlib

   HILFE:
     Ausfuehren dieses Programms: python PROGRAMMNAME
     Dieses Programm ist in der Sprachversion 2 von python geschrieben.

     numpy-Objekte werden mit prefix "np." aufgerufen
     matplotlib-Objekte werden mit prefix "plt." aufgerufen

.. author:: Günter Quast <g.quast@kit.edu>
     für den Kurs Computergestützte Datenauswertung
'''
#-----------------------------------------------------------

    . . .
```

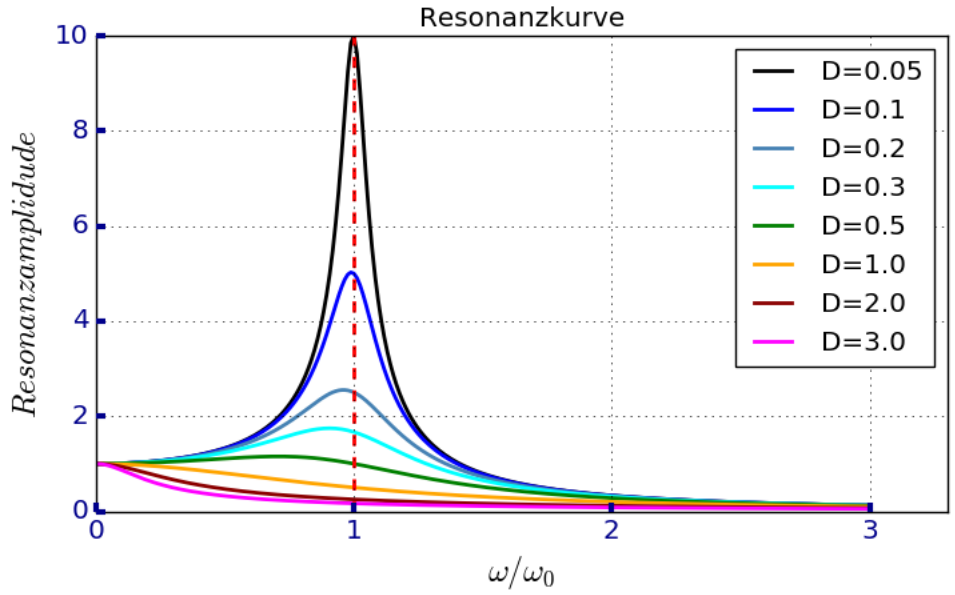
PlotBeispiel.py

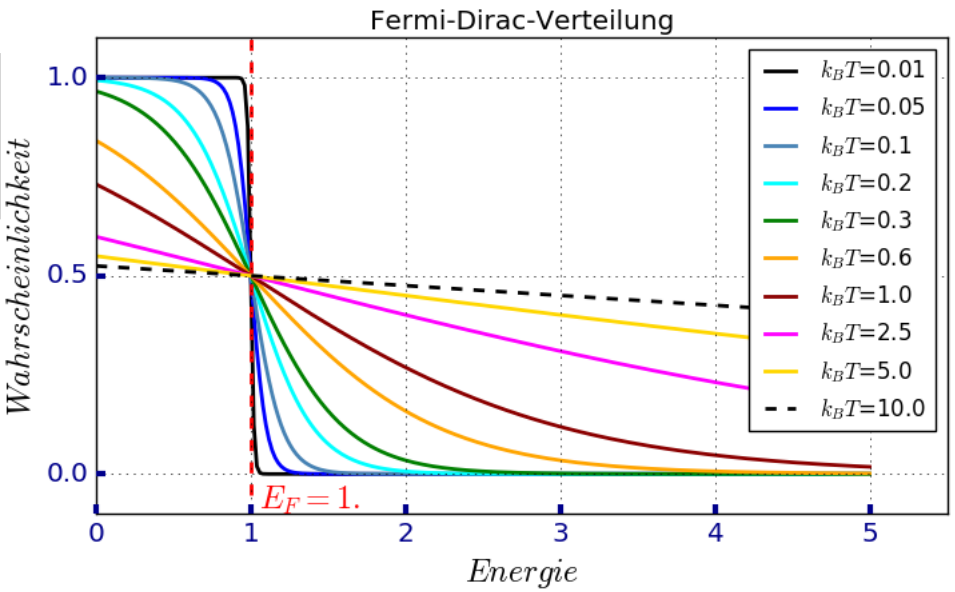


noch eine (flexiblere) Vorlage

```
# -*- coding: utf-8 -*-
#      diese Zeile legt die Codierung von Umlauten fest
##############################################

# script FunctionPlotter.py
'''
   Funktionsdarstellung mit matplotlib

.. author:: Günter Quast <g.quast@kit.edu>
     für den Kurs Computergestützte Datenauswertung
'''
#-------------------------------------------------------------


     . . .
```

siehe FunctionPlotter.py
nutzt myPlotstyle.py

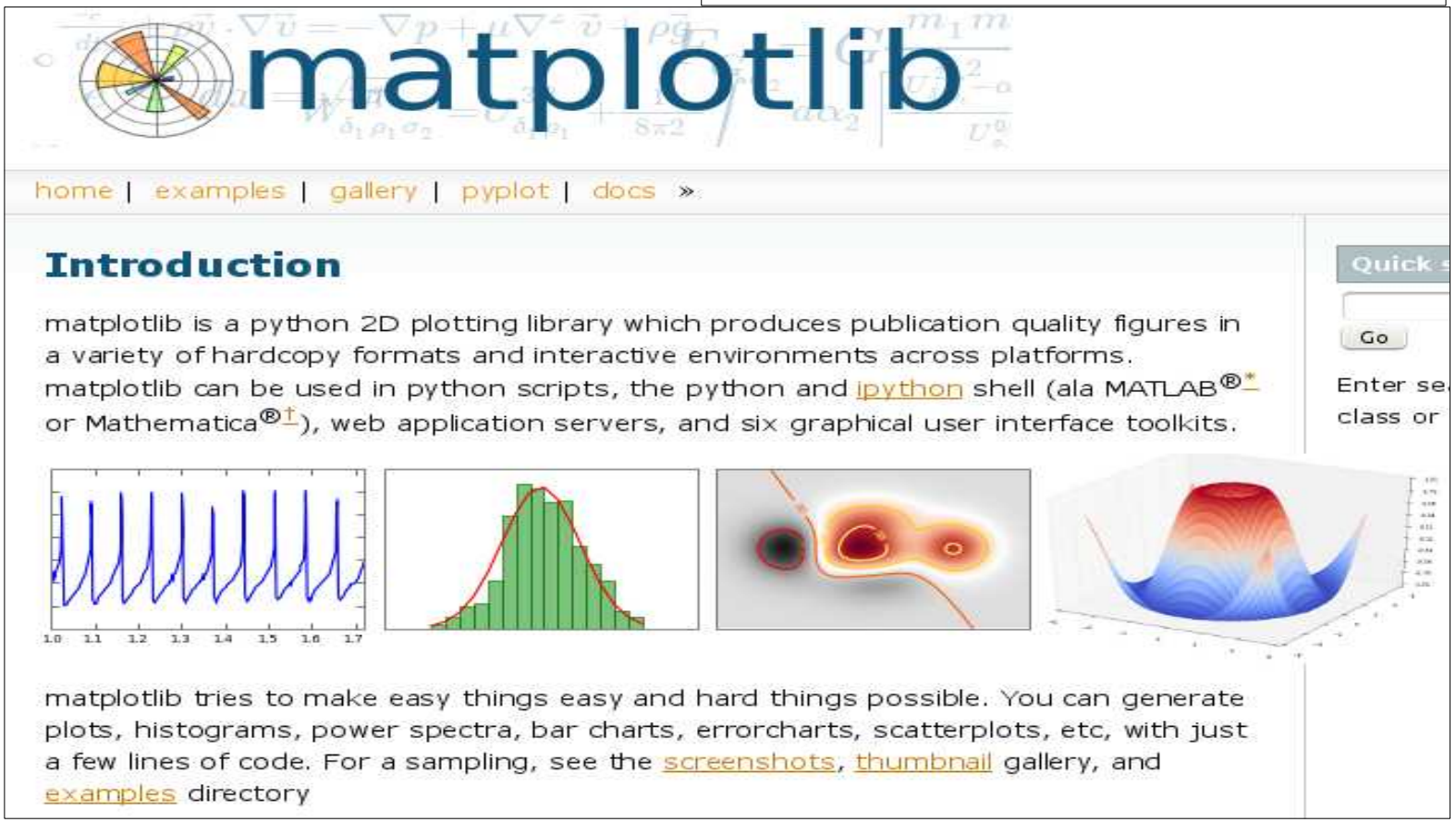




noch mehr Beispiele

animate_wave.py
animated_Gauss.py
throwCoin.py

Beispiele unter http://matplotlib.org



Auswertung einer Längenmessung

Daten in Text-Editor eingeben:

```
# Messung 28. Apr. 2016
# h[cm]
xx.xx
yy.yy
....
```

liveData.dat

Mit kleinem *Python*-script darstellen und auswerten →

s. auch liveDisplay.py
(in der Vorlesung verwendet)

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

#read columns from file:
infile="Data.dat"
A=np.loadtxt(infile,unpack=True)

print A # check input

# visualize as histogram  (with matplotlib)
fig=plt.figure(figsize=(10,10))
plt.hist(A,100)
plt.title('Distribution of Data')

print "Mittelwert:        %.3f" % np.mean(A)
print "Standardabweichung: %.3f" % np.std(A)
print "Unsicherheit auf Mittelwert: %.3f" \
      % (np.std(A)/np.sqrt(len(A)))

plt.show()
```

read_array.py

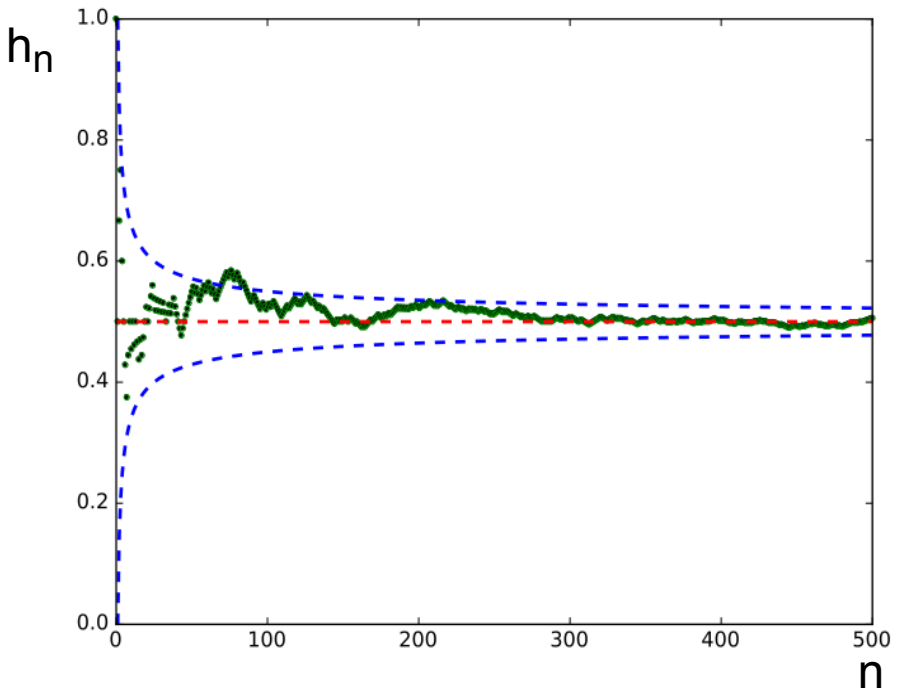
Wahrscheinlichkeit: Kopf oder Zahl ?

Beispiel Münzwurf:

Wahrscheinlichkeit für Kopf: $p_K=0.5$
Wahrscheinlichkeit für Zahl: $p_Z=1-p_K=0.5$

i

Führe *N=1, .., N* Computer-Experimnte durch, berechne jeweils die Häufigkeit $h_n = N_K(n) / n$



throwCoin.py

```
# throw a coin N times
import numpy as np
N=500
f=[ ]
Nh=0
for n in range(N):
  if np.random.rand()>0.5:
    Nh+=1
  f.append(float(Nh)/(n+1.))
```

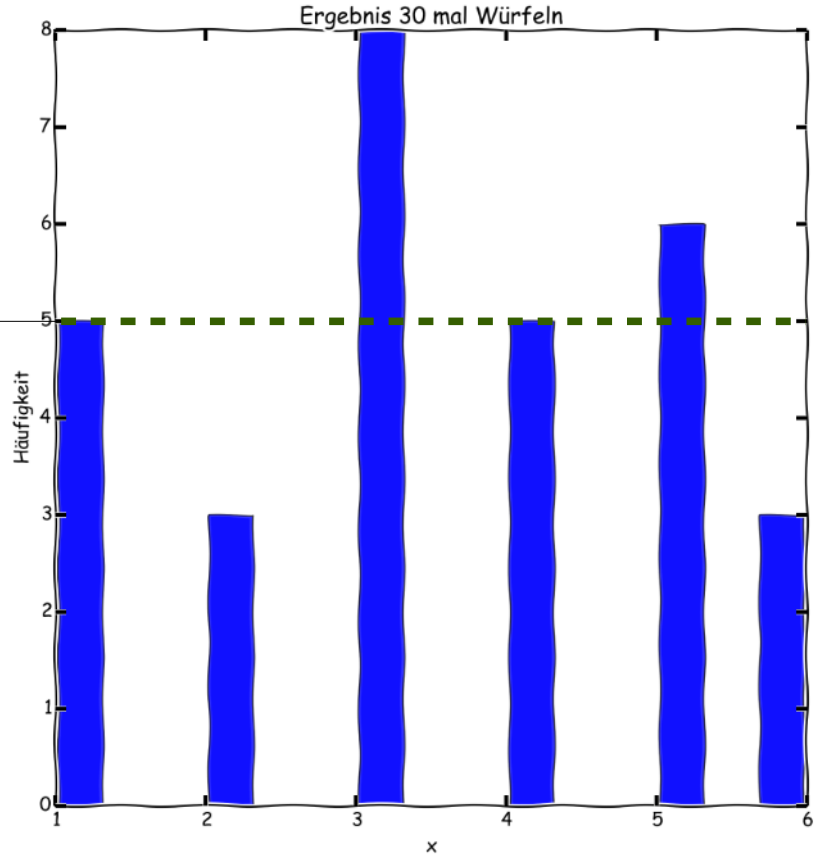
Häufigkeit
nähert sich der
Wahrscheinlichkeit an:

$$h_n \rightarrow p_K(n)$$

Häufigkeitsverteilung: diskret

Beispiel **Würfeln**:
Häufigkeit der Augenzahlen
1, 2, …, 6 bei 30 Würfen

Erwartete Verteilung



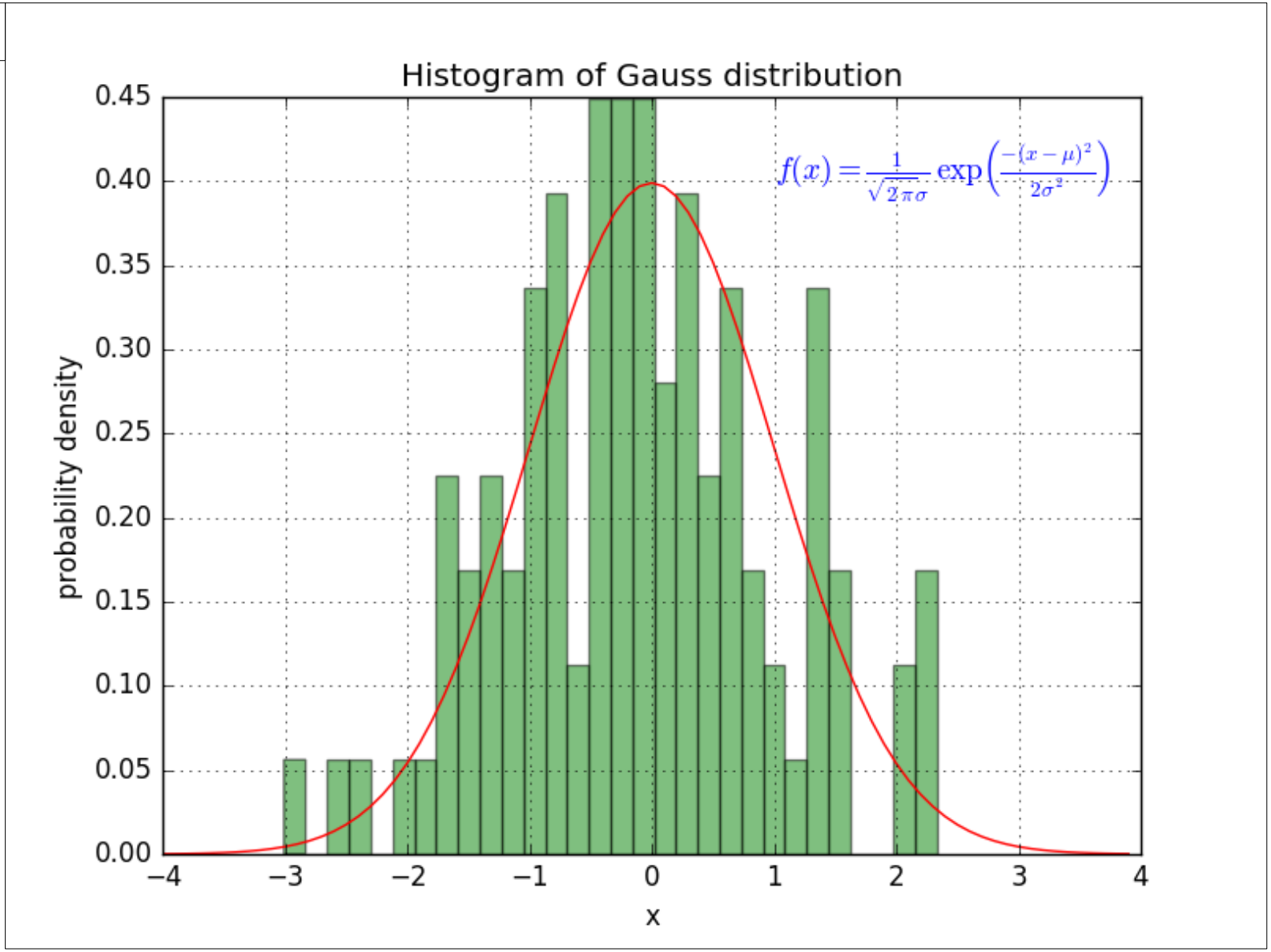
throwDice.py

```
# example to produce random numbers
#    as obtained by throwing dice
import numpy as np

ndice=1       # number of dice
nthrow=30     # how often to throw

numbers=[ ]
for i in range(nthrow):
  r=0
  for j in range(ndice):
    r += int(6*np.random.rand())+1
  numbers.append(r)
```

Häufigkeitsverteilung mit Verteilungsdichte

Häufigkeitsverteilung (Histogramm) mit Verteilungsdichte (rote Kurve) und Beschriftung

Gauss.py



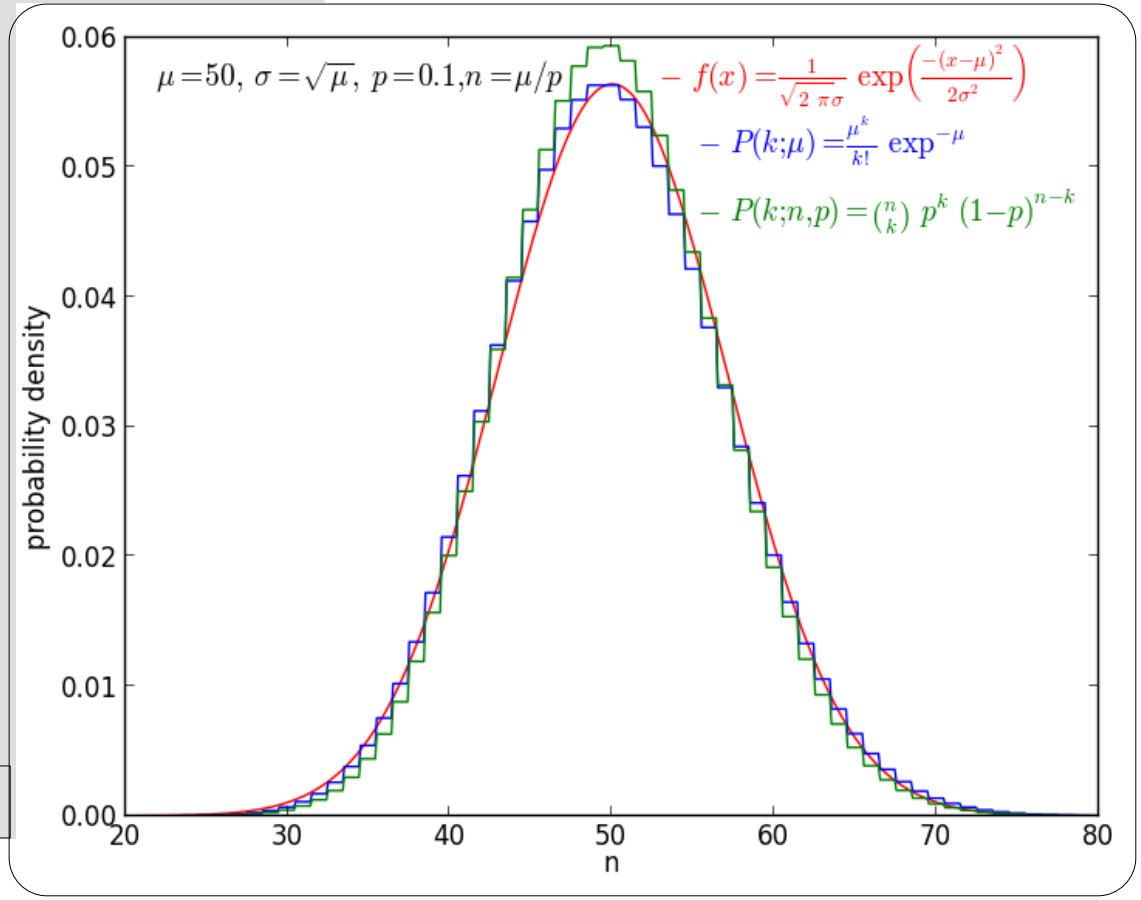
Binomial ↔ Poisson ↔ Gauss - Verteilungen

- für große n nähert sich die Poisson-Vert. einer Gauß-Verteilung an
- für kleine p und große n nähert sich die Binomial- der Poissonverteilung an mit μ= np

```
import numpy as np, matplotlib.pyplot as plt, scipy.special as sp

def fGauss(x,mu=0.,sigma=1.):  # Gauss distribution
    return (np.exp(-(x-mu)**2/2/sigma**2)/np.sqrt(2*np.pi)/sigma)

def fPoisson(x,mu): # Poisson distribution
  k=np.around(x)
  return (mu**k)/np.exp(mu)/sp.gamma(k+1.)

def fBinomial(x,n,p): # Binomial distribution
  k=np.around(x)
  return sp.binom(n,k) *p**k *(1.-p)**(n-k)

#-----------------------------------------------------------
# plot basic distributions
mu, sig = 50., np.sqrt(mu)
x = np.arange(20, 80., 0.1)
plt.plot(x, fGauss(x,mu,sig), 'r-')
plt.plot(x, fPoisson(x,mu), 'b-')
p, n = 0.1, mu/p
plt.plot(x, fBinomial(x,n,p), 'g-')

#  ( ... )  # produce nice text
plt.show()
```

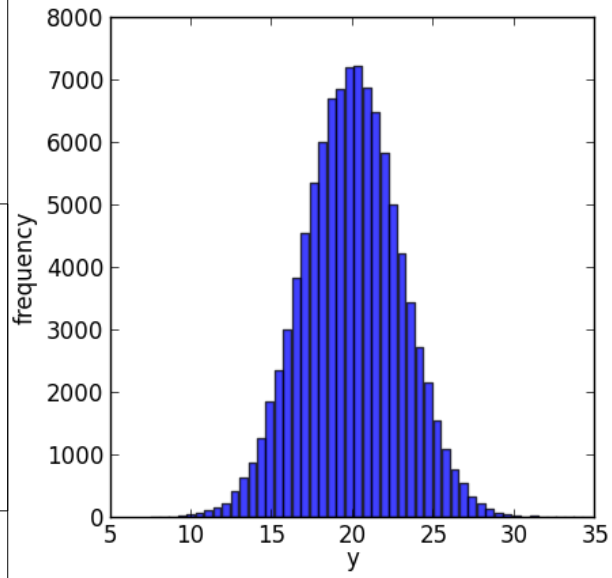
basicDistributions.py

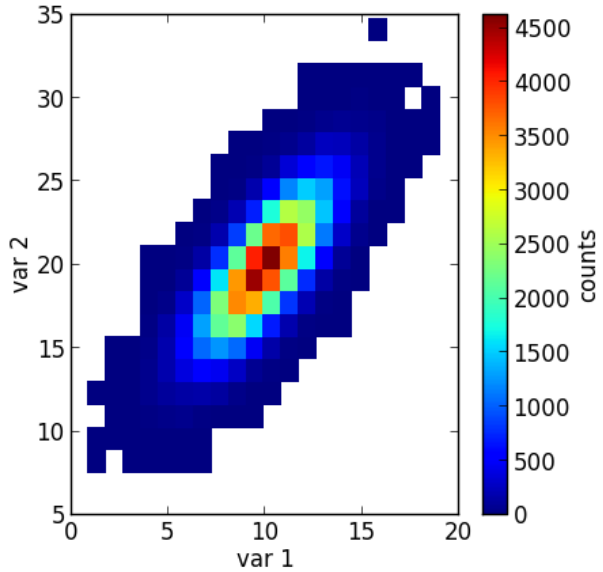
← einige spezielle Funktionen in `scipy.special` [*n! = gamma(n+1)*]



2d-Histogramme in *matplotlib*

hist2d statistics:
<x> = 10.0093
<y> = 19.9881
var_x = 4
var_y = 9.1
cov = 4.5, cor = 0.75





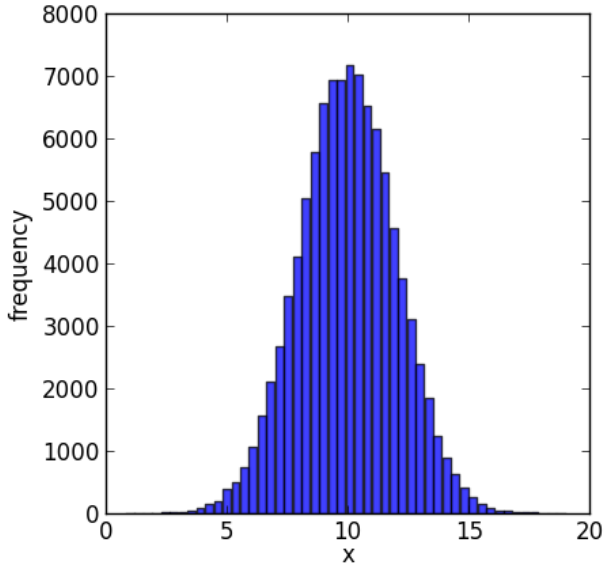
python / numpy / matplotlib:

```
numpy.histogram(),
numpy.histogram2d()
matplotlib.pyplot.bar(),
matplotlib.pyplot.pcolormesh()
```

siehe auch:

```
matplotlib.pyplot.hist()
matplotlib.pyplot.hist2d()
```

Beispielscript `Histogram.py`



Anpassung mit einem Parameter

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

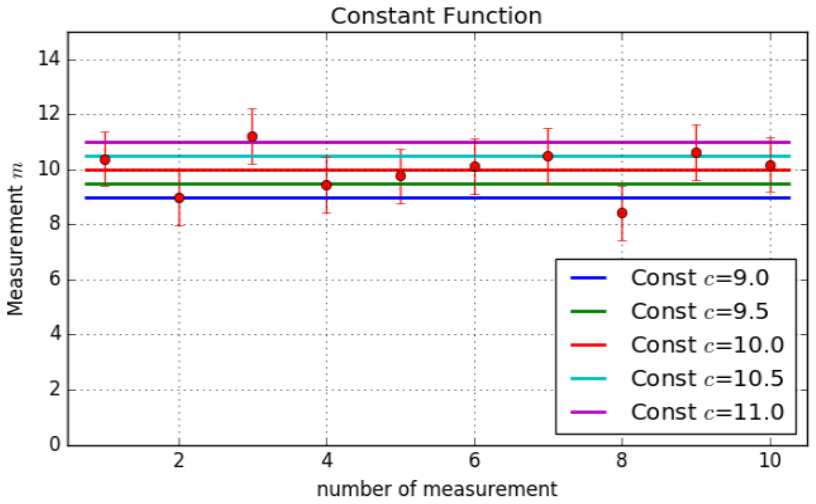
$$S(c) = \sum_{i=1}^{10} \frac{(y_i - c)^2}{\sigma^2}$$

analytisch:

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

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

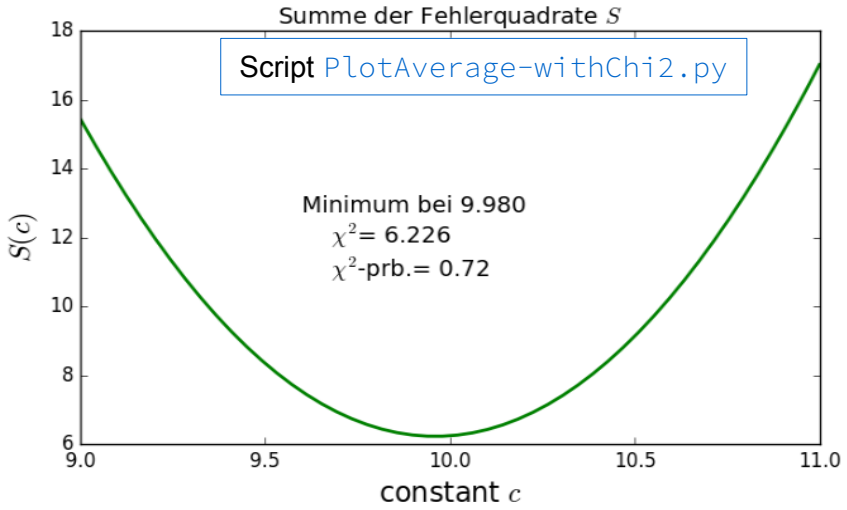
identisch zum „Mittelwert“



„numerisch“:

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

berechnen und
grafisch darstellen



Beispielcode: numerische Anpassung 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.

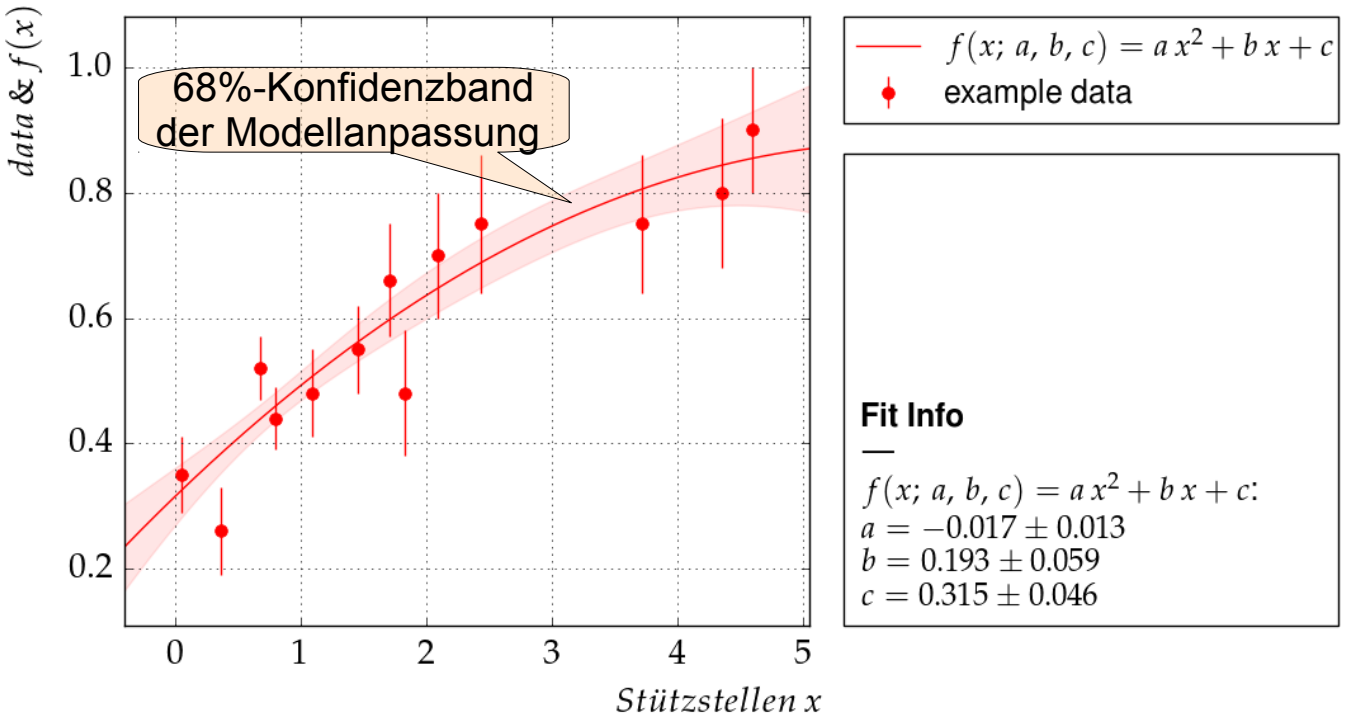
Zuverlässige Konvergenz zum globalen Minimum bei nicht-linearen Problemstellungen erfordert meist das Setzen geeigneter Startwerte!

Beispiel mit dem python-Framework

„kafe“:

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

siehe Script
`fitexample_kafe.py`

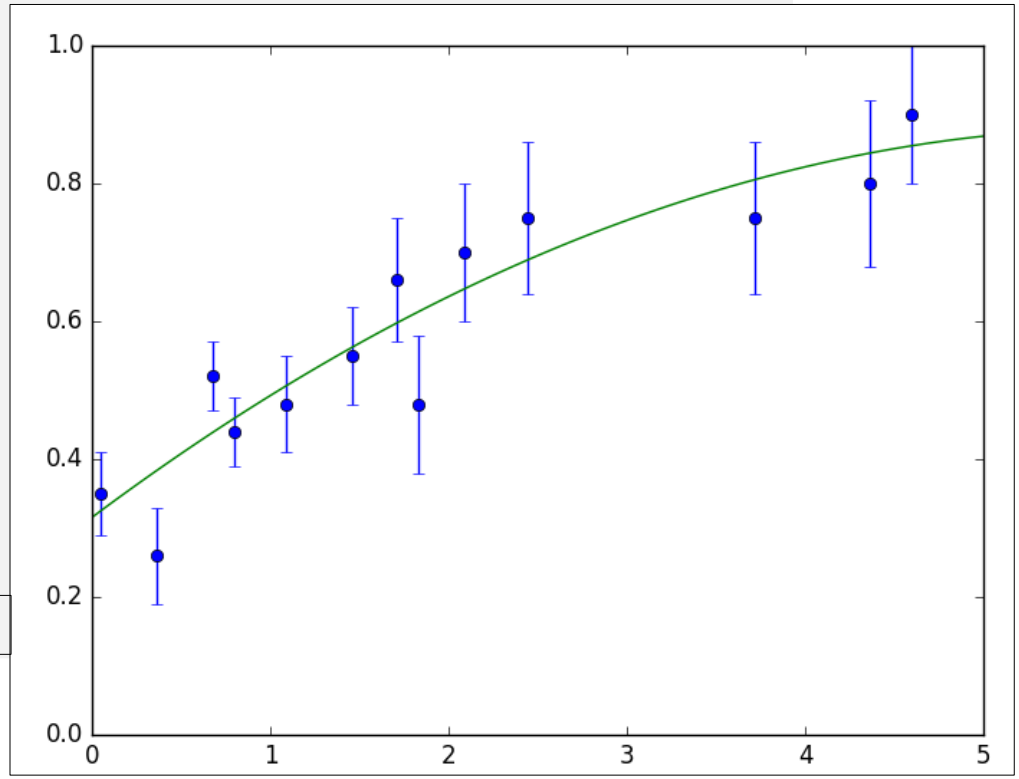


Beispielcode: numerische Anpassung (scipy)

Tool curve_fit aus scipy.optimize
zur χ^2-Anpassung und numerischen Optimierung

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.optimize import curve_fit
# ------------------------------------------------------------------------------

def chi2_fit():
#-- set data points
  xm = np.array([.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 = np.array([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 = np.array([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])

#-- least-squares fit with scipy.optimize.curve_fit
  p, cov = curve_fit( poly2, xm, ym, sigma=ye, absolute_sigma=True )
  print "Fit parameters:\n", par
  print "Covariance matrix:\n", cov
#-- plot data and fit result
  xp = np.linspace( 0., 5., 100 )
  ffit = poly2(xp, p[0], p[1], p[2])
  plt.errorbar(xm, ym, yerr=ye, fmt='o')
  plt.plot( xp, ffit, '-' )
  plt.xlim( 0, 5 )
  plt.ylim( 0, 1 )
  plt.show()

#-- define fit function
def poly2(x, a=1., b=0., c=0.):
  return a * x**2 + b * x + c

if __name__ == '__main__':  # ------------
  chi2_fit()
```

curvefit_example.py



Unsicherheiten mit „Toy-MC“

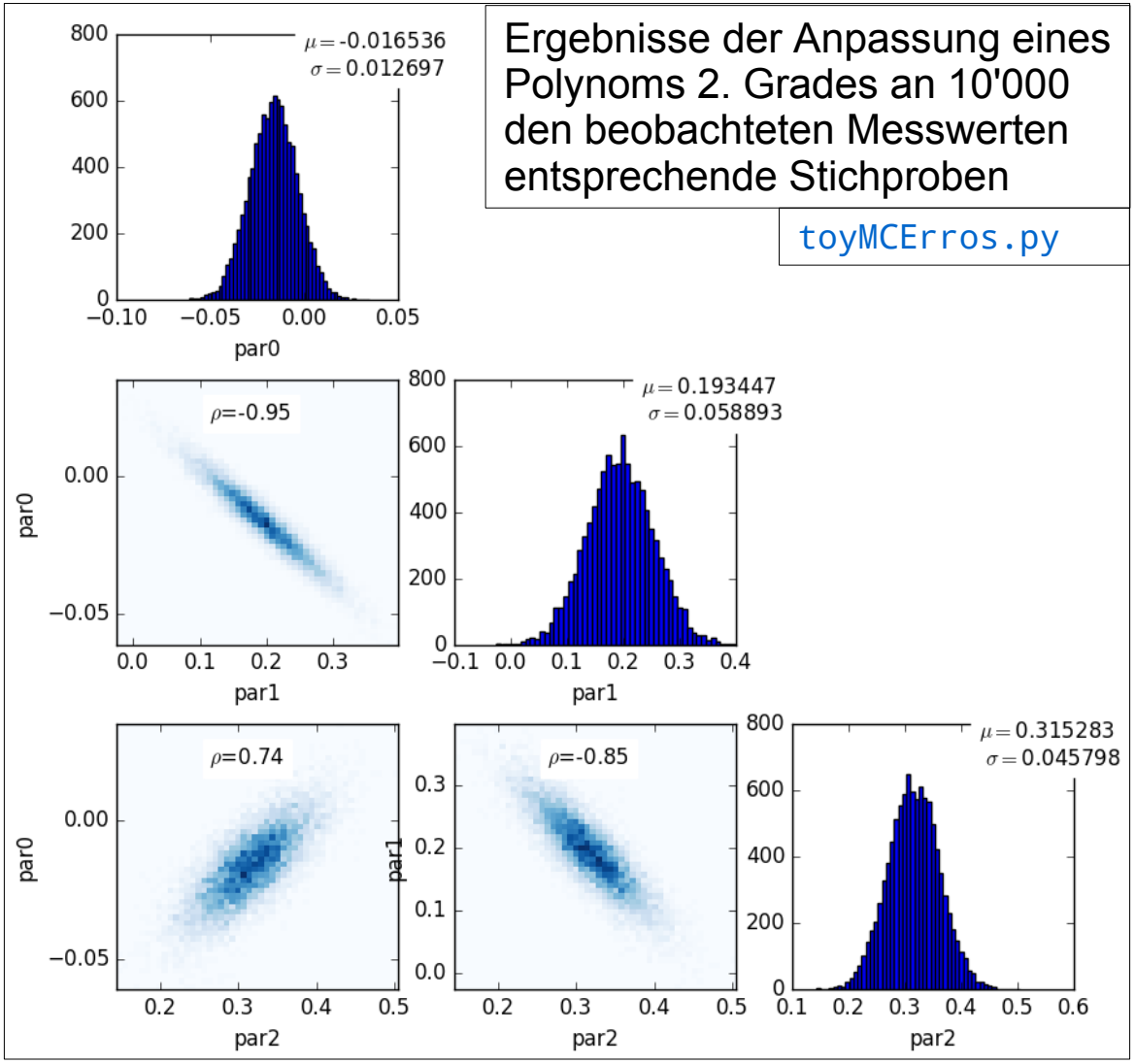
Die Unsicherheiten können ganz allgemein bestimmt werden, indem man die Anpassung für viele Stichproben wiederholt und die Verteilungen der Parameterwerte analysiert.

Auf diese Weise lassen sich
- Parameterunsicherheiten
- Korrelationen der Parameter

bestimmen.

Das Verfahren ist sehr aufwändig, aber exakt !

Im Zweifelsfall wird die Exaktheit statistischer Verfahren mit solchen **„Toy – Monte Carlos“** untersucht.

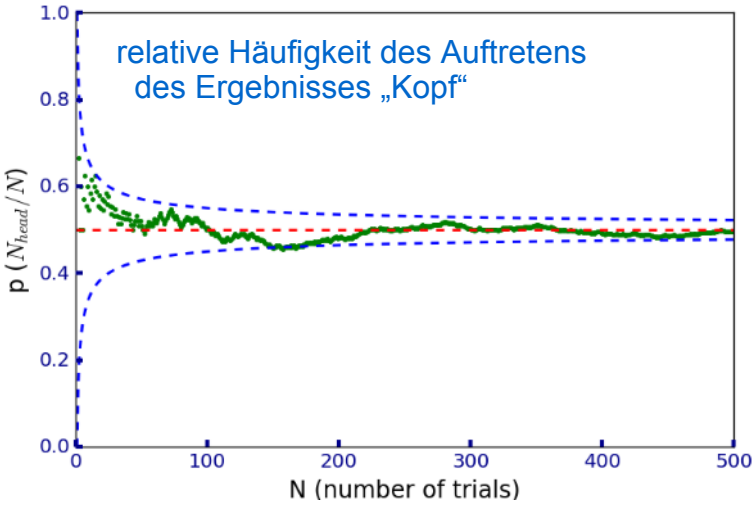


Ergebnisse der Anpassung eines Polynoms 2. Grades an 10'000 den beobachteten Messwerten entsprechende Stichproben

`toyMCErros.py`

Beispiel: Likelihood beim Münzwurf

Erinnerung: Binomialverteilung beim Wurf einer Münze:



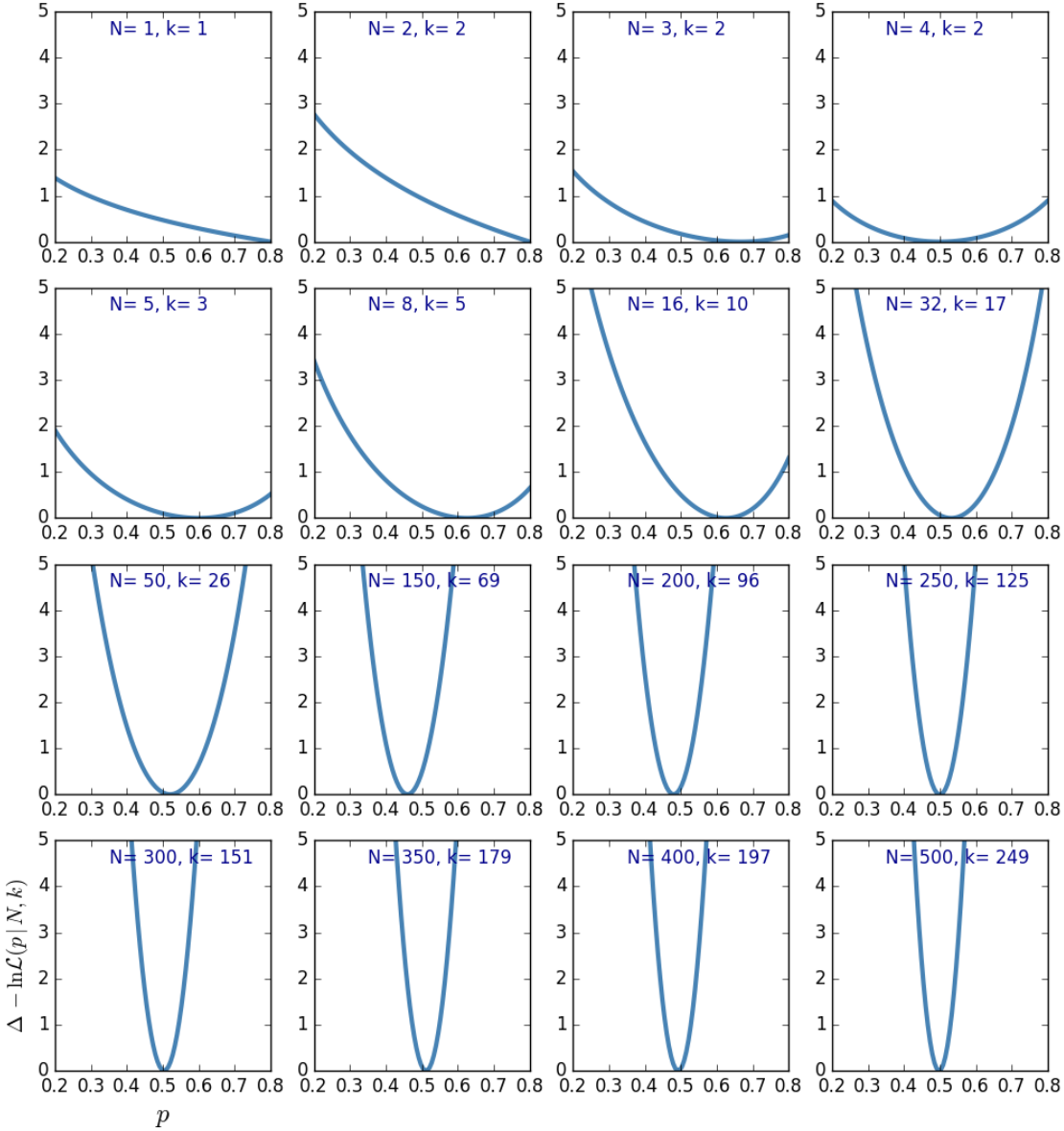
Für einige der Ergebnisse aus der Reihe von Münzwürfen oben ist nebenan die jeweilige Likelihood-Funktion gezeigt:

$$-\ln\mathcal{L}(p|k;N) = -\ln\left(B(k;p,N)\right)$$
$$= -\ln\left(\binom{n}{k} p^k (1-p)^{n-k}\right)$$

$-\ln\mathcal{L}(p|k;N)$ ist eine Funktion des Parameters *p* für gegebene Beobachtung *(N, k)*

$$-\ln\mathcal{L}(p|k;N) = -\ln\left(B(k;p,N)\right)$$

Skript `nlLCoin.py`



Mit zunehmender Zahl an Würfen wird der Parameter *p* durch die Likelihood-Funktion immer genauer eingegrenzt

[** Beispiel: Likelihood für Poisson-Prozess]

Ein typisches Beispiel für die Anwendung der negLog$\mathcal{L}$-Methode sind kleine Stichproben von Poission-verteilten Beobachtungen mit kleinem Erwartungswert:

Stichprobe:

Poisson-verteilte Zufallszahlen

$\{k_i\}$= { 0, 2 , 4 , 4 , 2 ,1, 1, 1, 1 }

$$P(k;\lambda) = \frac{\lambda^k}{k!}\mathrm{e}^{-\lambda} \Rightarrow$$

$$-\ln\mathcal{L}(\lambda|k_i, i = 1, ..., n) =$$

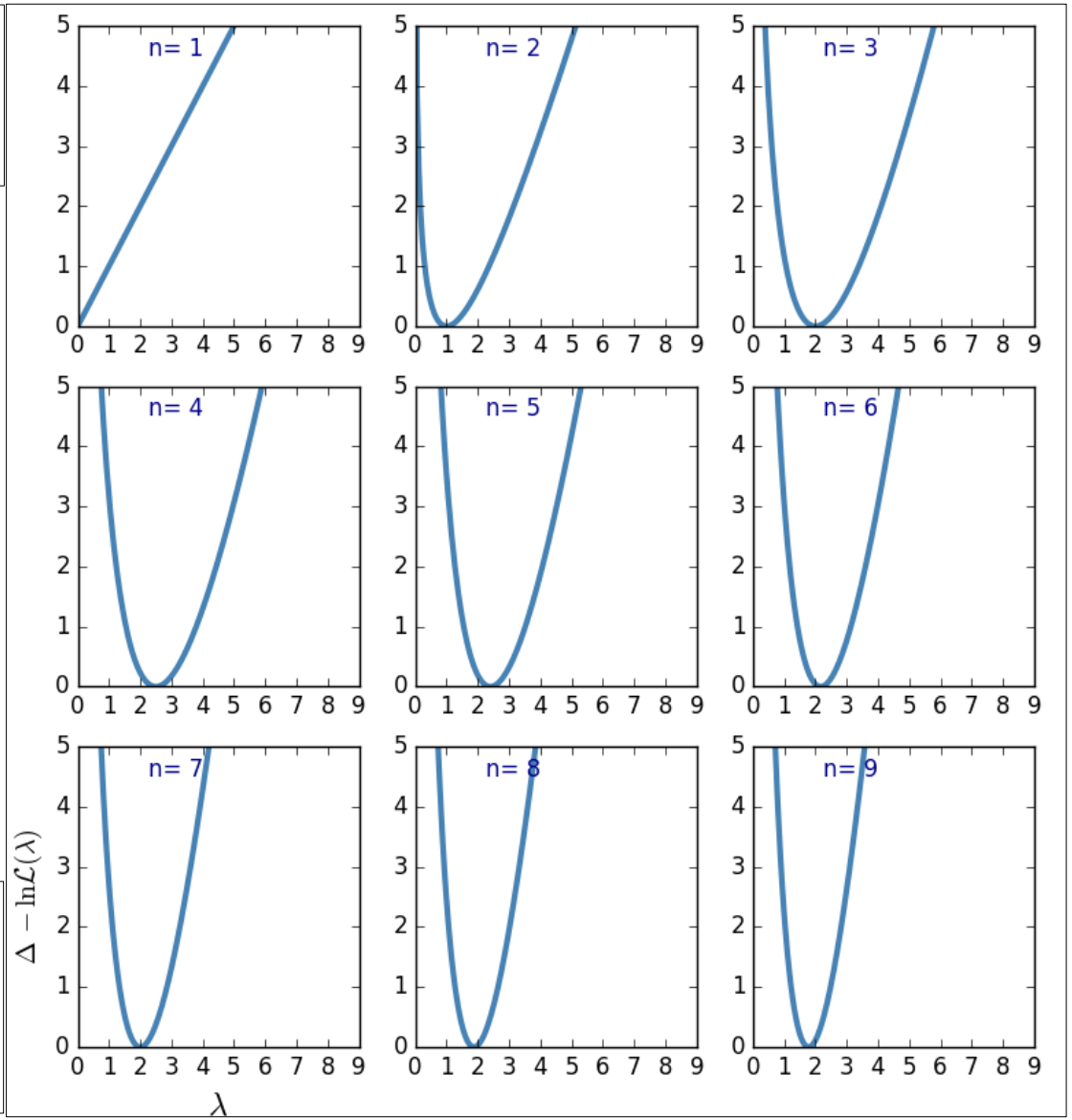
$$-\sum_{i=1}^{n}(-\lambda + k_i \ln\lambda - \ln(k_i!))$$

allg. Implementierung ist etwas anders, s. Codebeispiel nlLExample.py

Darstellungen von

$-\ln \mathcal{L}(\lambda \mid \{k_i, i \le n\})$

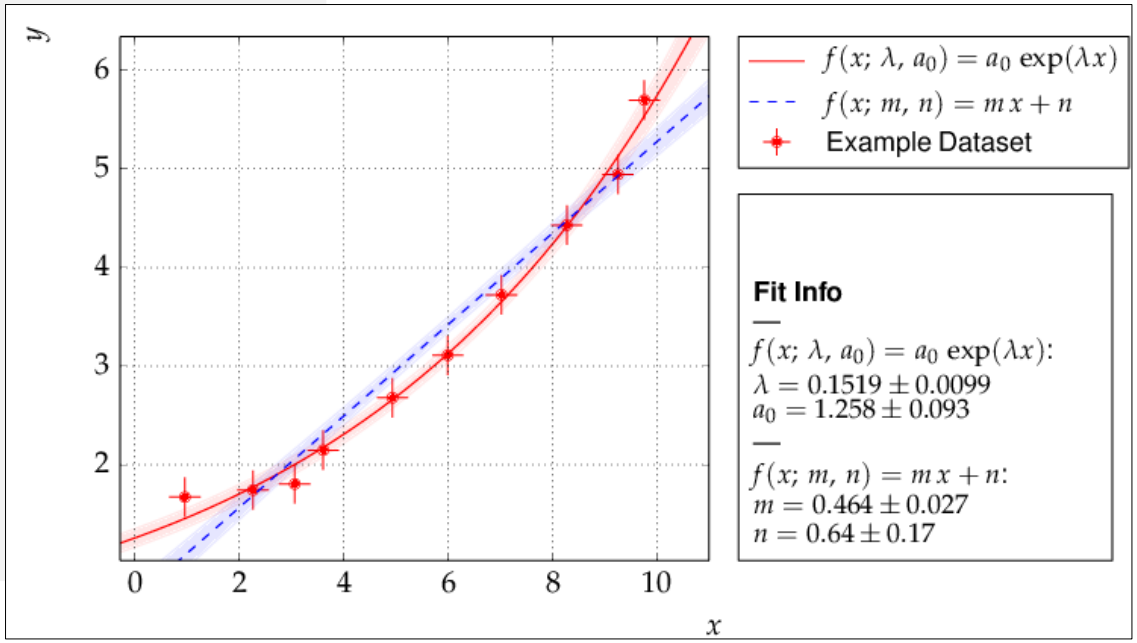
für die jeweils *n* ersten Elemente der Stichprobe



kafe example1 : Vergleich von zwei Modellen

```
# import everything we need from kafe
from kafe import *
# additionally, import the two model functions we want to fit:
from kafe.function_library import linear_2par, exp_2par
#############
# Load the Dataset from the file
my_dataset = Dataset(input_file='dataset.dat',
          title="Example Dataset")
### Create the Fits
my_fits = [ Fit(my_dataset, exp_2par),
            Fit(my_dataset, linear_2par) ]
### Do the Fits
for fit in my_fits:
   fit.do_fit()
### Create the plots, save and show output
my_plot = Plot(my_fits[0], my_fits[1])
# show data only once (it's the same data
my_plot.plot_all(show_data_for=0) )
my_plot.save('plot.pdf')
my_plot.show()
```

```
linear_2par
chi2prob 0.052
HYPTEST  accepted (CL 5%)
exp_2par
chi2prob 0.96
HYPTEST  accepted (CL 5%)
```



Daten-getriebene Anpassung

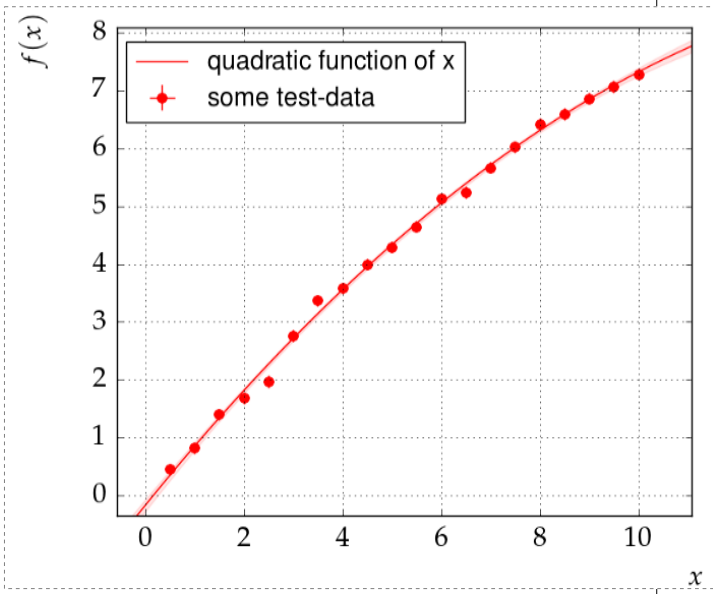
Die Methode kafe.file_tools.build_fit_from_file() ermöglicht es, einfache Anpassungen vollständig über eine Datei zu steuern:

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
# show everything on screen
thePlot.show()
```

Datei data.fit

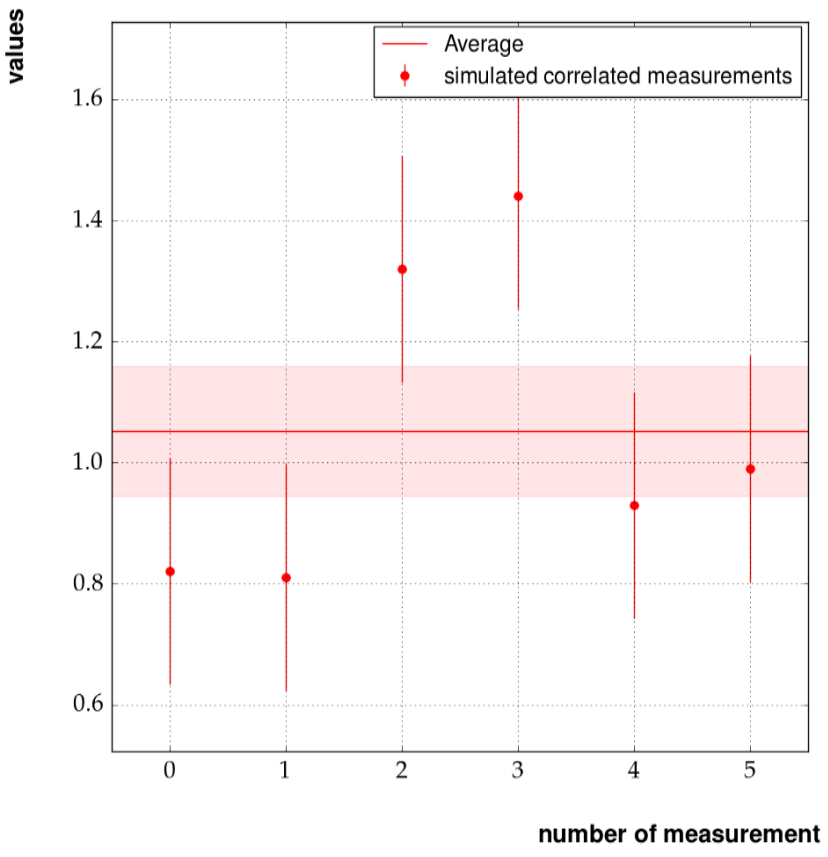
```
# example showing fit driven by input file
# ----------------------------------------
# this file is to be parsed with
#         kafe.file_tools.buildFit_fromFile()

#  Meta data for plotting
*BASENAME quadraticFitExample
*TITLE  some test-data
*xLabel $x$
*xUnit
*yLabel $f(x)$
*yUnit

# the data
*xData
  0.50 0.1
  1.00 0.1
   - - - -
  9.50 0.1
 10.00 0.1

*yData
  0.45 0.1
  0.82 0.1
   - - - -
  7.07 0.1
  7.29 0.1

*FITLABEL quadratic function of x
*FitFunction
def fitf(x,a,b,c):
~~return a*x**2 + b*x +c

*InitialParameters    # set initial values and range
0.   0.5
1.   0.5
0.   0.5
```



gleiches Skript, anderer Input: Mittellung korrelierter Messungen

Gleiches Skript (kafe_fit-from-file.py) mit anderen Eingabedaten

(Mess-)Werte mit Unsicherheit und Kovarianzmatrix (angegeben als Wurzel aus den Nebendiagonalelementen)



aveCorrDat.fit

```
# Simulated correlated measurements
#  - common error between pairs of measurements
#  - common error of all measurements
# ----------------------------------------------------------------

*BASENAME averageCorrDat
*TITLE simulated correlated measurements
*xLabel number of measurement
*yLabel values

#*xData # commented out, as not needed for averaging
*yData_SCOV
#  val    err      syst     sqrt. of cov. mat. elements
   0.82   0.10     0.15
   0.81   0.10     0.15     0.15
   1.32   0.10     0.15     0.   0.
   1.44   0.10     0.15     0.   0.   0.15
   0.93   0.10     0.15     0.   0.   0.    0.
   0.99   0.10     0.15     0.   0.   0.    0.   0.15
# common (correlated) error for all
*yAbsCor 0.05

# python code of fit function
*FITLABEL Average
*FitFunction
@ASCII(expression='average')
@LaTeX(name='f', parameter_names=('m',), expression='m')
@FitFunction
def fitf(x, m=1.): # fit an average
~~~~return m

*InitialParameters
1. 1.
```