Karlsruher Institut für Technologie

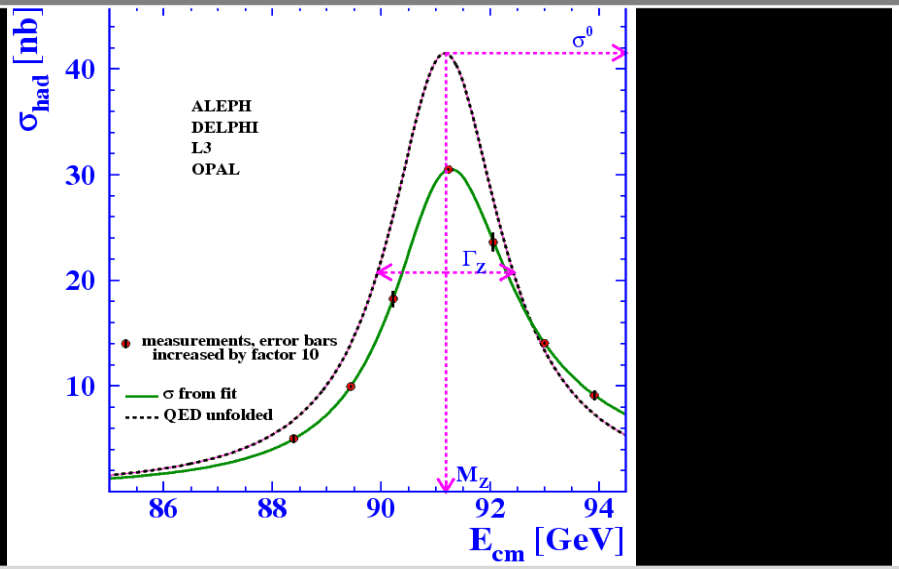
Vorlesung 05: Rechnernutzung in der Physik

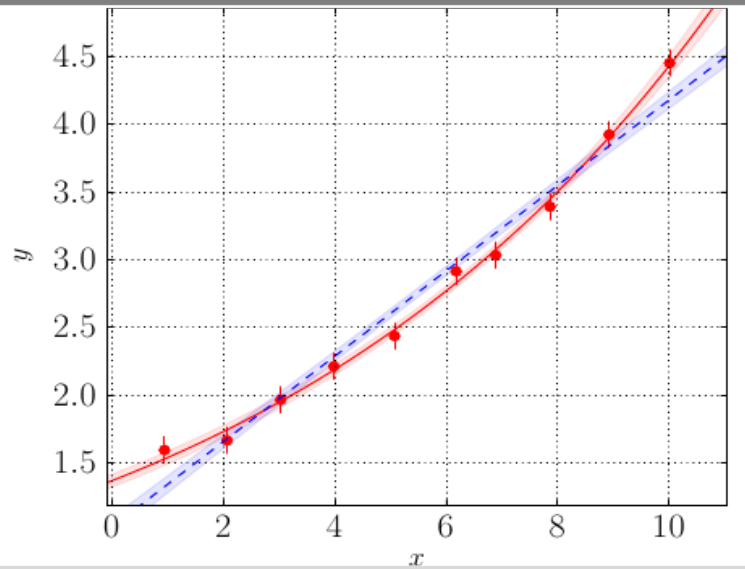
Modellanpassung mit der Likelihood-Methode

Günter Quast

Fakultät für Physik
Institut für Experimentelle Kernphysik

WS 2023/24





Maximum Likelihood

Maximum-Likelihood-Methode: mächtiges Verfahren zur Parameterschätzung

Likelihood-Funktion $\mathcal{L}$: Produkt der Werte der Wahrscheinlichkeitsdichte P_i für ***n* unabhängige Beobachtungen** (bzw. Messungen) x_i :

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

hängt nach Beobachtung der Daten x_i nur noch von den Parametern ***a*** ab !

Maximum-Likelihood (ML) Prinzip:

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

Technische und theoretische Gründe:

Minimiere den **negativen Logarithmus** der Likelihood-Funktion:

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

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

Bedingung für Optimum

Zusammenfassung: Eigenschaften der ML-Schätzung

Maximum-Likelihood-Schätzer

- ist asymptotisch effizient, **d. h. hat minimale Varianz**
- ist asymptotisch unverzerrt, d.h. $\hat{a} = a + b, \ \text{mit} \ b \to 0$ für große Stichproben
- ist invariant unter Parameter-Transformationen $\mathrm{a} \to \Theta(a)$:

$$\frac{\partial}{\partial \Theta}\mathcal{L}(a(\Theta)) \quad = \quad \frac{\partial}{\partial a}\mathcal{L}(a) \cdot \frac{\partial}{\partial \Theta}a(\Theta)$$

$$\frac{\partial}{\partial a}\mathcal{L}(a) = 0 \quad \Rightarrow \quad \frac{\partial}{\partial \Theta}\mathcal{L}(\Theta) = 0$$

dies erlaubt z.B.

Reduktion von Verzerrungen durch Parametertransformation

Aus Likelihood-Scan abgelesene Konfidenzintervalle sind ebenfalls invariant unter Parameter-Transformation !

(gilt nicht für Bestimmung der Unsicherheit aus 2. Ableitungen)

Übersicht: Methoden zur Bestimmung der 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:

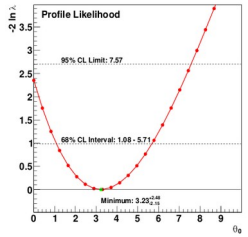
- Likelihood-Analyse:
  Ausnutzen der Cramer-Rao-Frechet-Grenze:

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 großer Stichproben)

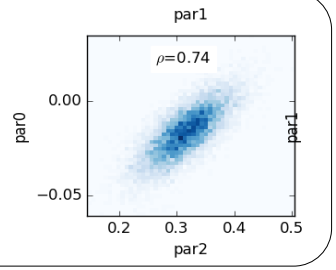
- Scan der (Profil-) Likelihood in der Nähe des Minimums

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



Bei Unklarheit oder sehr kleinen Stichproben:

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



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

Übungsblatt

Das **dritte Übungsblatt**
ist on-line !

Rechnernutzung in der Physik

Institut für Experimentelle Teilchenphysik
Institut für Theoretische Teilchenphysik
Prof. G. Quast, Prof. M. Steinhauser
Dr. A. Mildenberger, Dr. Th. Chwalek
Ilias Seite zum Kurs
WS 2023/24 – Blatt 03
Abgabe: Montag 27.11.2023 bzw. Dienstag 28.11.2023

Themen:

- Poisson-Likelihood
- Profile Likelihood

ggf. Problem mit nicht installiertem Paket *PhyPraKit*:

im Terminal des Jupyter-Notebooks eingeben: pip install --user PhyPraKit

Block 1: Statistik und Datenanalyse

Themen:

31.10. Grundlagen der Statistik (Wiederholung CgDA);
python & friends, jupyter-Tutorials

07.11. Monte Carlo-Methode als numerisches Hilfsmittel
MC-Tools in PhyPraKit

14.11. Parameterschätzung mit der Maximum-Likelihood-Methode
Hilfsfunktionen in PhyPraKit & kafe2

21.11. Maximum Likelihood (2) und numerische Optimierung
Ensemble-Tests

28.11. Hypothesentests

Spezielle Problemstellungen:

- relative Unsicherheiten
- extern festgelegte/eingeschränkte Parameter
- Anpassung an Histogrammdaten
- Anpassung an (x,y) Daten
- Unsicherheiten in Abszissen-Richtung

Relative Unsicherheiten

Oft hängen die Unsicherheiten von den Messwerten ab, $\sigma_i(y_i)$

z.B. relative Fehler, $\sigma_i \to \hat{\sigma}_i = c\, y_i$,

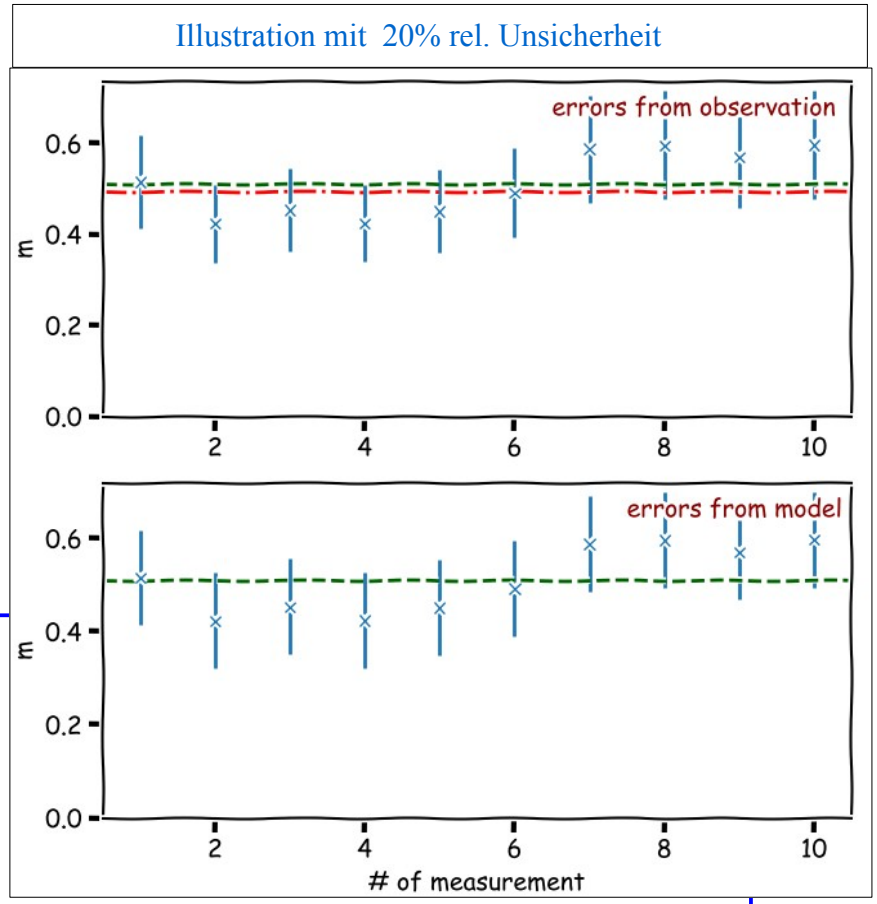
oder Poisson-Fehler, $\sigma_i \to \hat{\sigma}_i = \sqrt{y_i}$

Dann resultiert häufig eine **„verzerrte“ Anpassung**

Abwärtsfluktuationen der Messung
→ zu kleine Unsicherheit
→ überhöhtes Gewicht in der Anpassung

bzw.

Aufwärtsfluktuationen der Messung
→ zu große Unsicherheit
→ zu kleines Gewicht in der Anpassung



Abhilfe:

Unsicherheiten aus der Modellfunktion bestimmen,
dazu **Anpassung iterieren**

1.) mit den aus der Beobachtung geschätzten Unsicherheiten anpassen

2.) verbesserte Schätzung der Unsicherheit aus den nun bekannten Funktionswerten $f(x_i) \to \sigma_i\,(f(x_i))$ verwenden

[3.) zur Konvergenzkontrolle kann man ein drittes mal iterieren]

besser: in jedem Schritt der numerischen Optimierung Unsicherheiten anpassen

Achtung: Unsicherheiten sind parameterabhängig → nicht χ^2 sondern **Maximum Likelihood** verwenden !

Parameter mit Einschränkungen

Häufig tauchen in Anpassungsproblemen Parameter auf,
die auf externem Wissen bzw. externen Messungen beruhen:

- Naturkonstanten
- Materialeigenschaften
- Parameter der Messapparatur aus dem Datenblatt
- …

Es gibt mehrere **Möglichkeiten zur Behandlung**:

1. Parameter im Theorie-Modell als Konstante enthalten,
   für Parameter ohne bzw. mit vernachlässigbar kleiner Unsicherheit

2. gut bekannte Parameter, deren Einfluss studiert werden soll, sind im Theorie-Modell als freie Parameter enthalten (analog zu den anzupassenden Parametern), können in der Anpassung festgehalten, aber auf verschiedene Werte gesetzt werden (Bsp. kafe: Methode `fix_parameters()`)

3. nur ungenau bekannt Parameter sind im Theorie-Modell enthalten und werden auch in der Anpassung frei gelassen, aber **als weiterer „einschränkender Messwert“** (engl. constraint) **hinzugefügt**

also: für einen auf $c = c_0 \pm \sigma_c$ eingeschränkten Parameter $\mathcal{L} \to \mathcal{L}' = \mathcal{L} + \frac{(c - c_0)^2}{{\sigma_c}^2}$

Konfidenzband um Modell

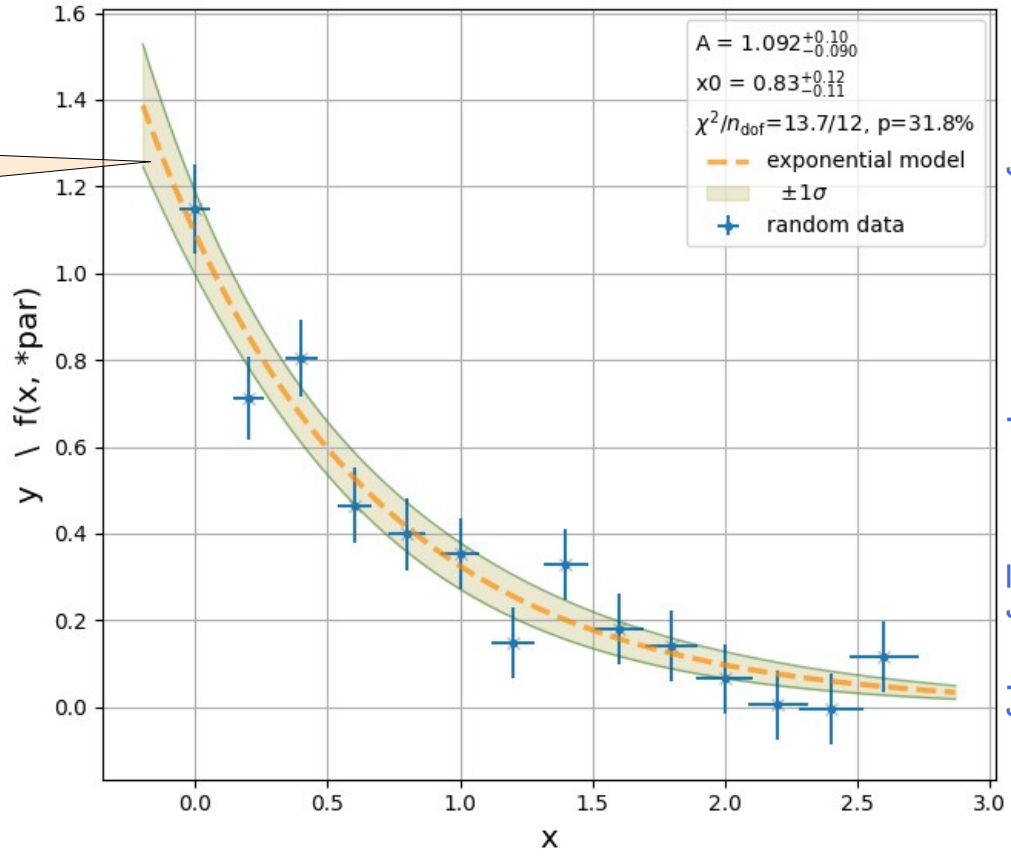
Eine Anpassung liefert Schätzwerte für die Parameter $\vec{\hat{a}}$

sowie ihre Unsicherheiten in Form der Kovarianzmatrix $\mathbf{V}_{\hat{a}}$

Mit Hilfe der **linearen Fehlerfortpflanzung** lässt sich die Unsicherheit auf das angepasste Modell bestimmen:

$$\sigma_f(x, \vec{\hat{a}}) = \left(\nabla_a f(x, \vec{\hat{a}})\right) \mathbf{V}_{\hat{a}} \left(\nabla_a f(x, \vec{\hat{a}})\right)^T$$

±1σ-Band der Modellanpassung



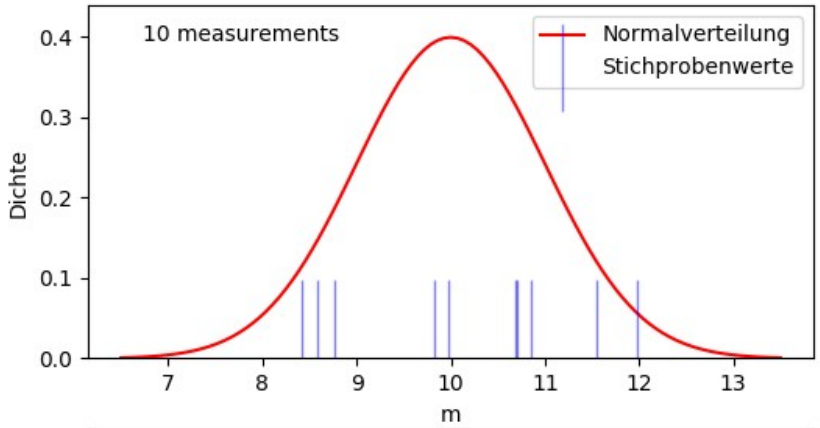
PhyPraKit//examples/test_xyFit.py

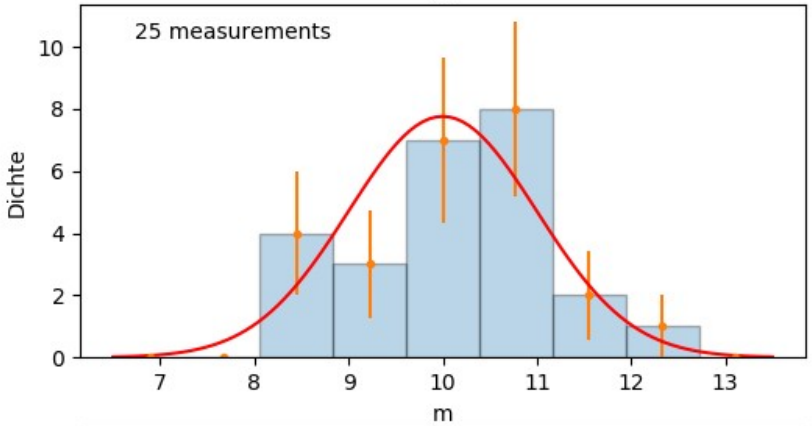
Übliche Darstellung:

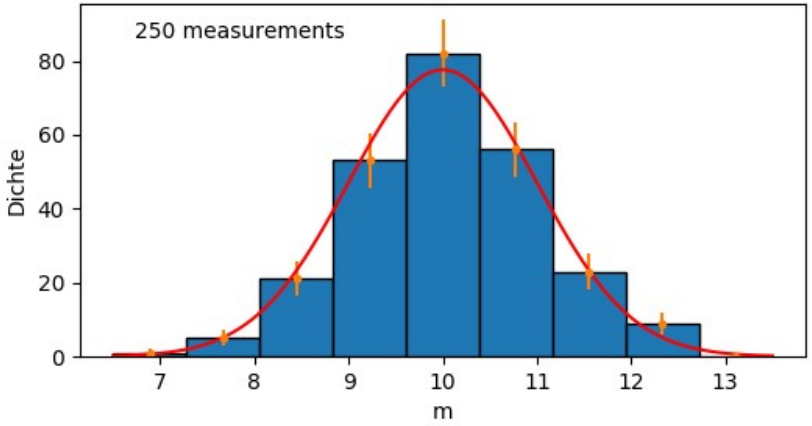
helles **±1σ-Band**

um die Zentralwerte des Modells

Anpassen von Funktionen an Histogramme







Wenige Messungen:

→ nur negLog$\mathcal{L}$- Anpassung sinnvoll

Histogramm als Näherung der Verteilung:

$$N_i^t = \int_{x_i - b/2}^{x_i + b/2} f(x;a)dx \quad \simeq b \cdot f(x_i;a)$$

– Auflösungsverlust durch Binning

– Unsicherheiten sind Poisson-verteilt

$$\sigma_{N_i} = \sqrt{N_i}$$

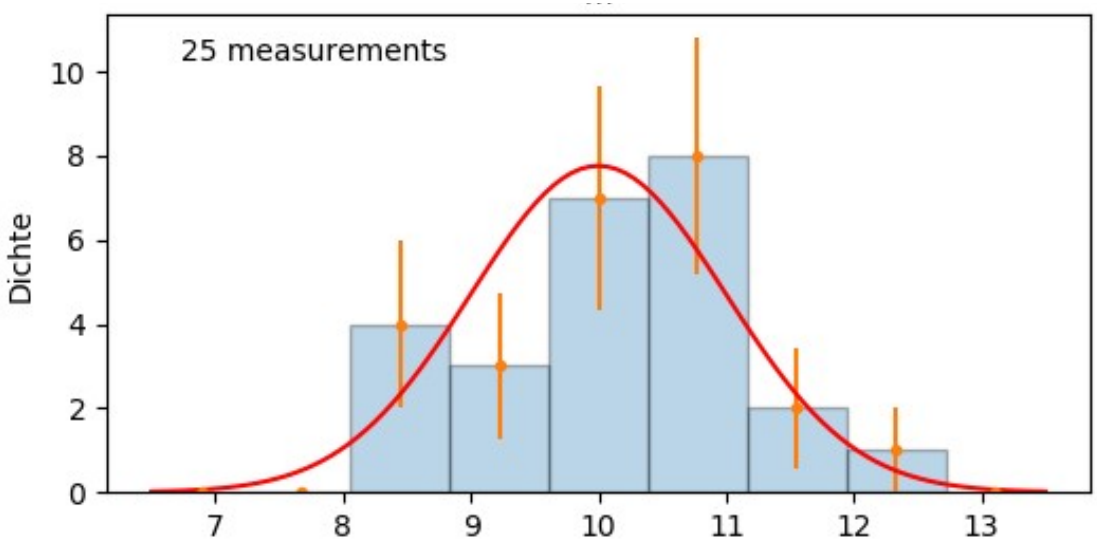
„Hinreichend viele“ Messungen:

Poissonverteilung → Gaußverteilung

– χ^2- Anpassung wird möglich

mit $\chi^2 = \sum_i \left(\frac{N_i - N_i^t}{\sqrt{N_i}} \right)$

Funktionsanpassung an Histogramme: **Binned -ln$\mathcal{L}$ - Fit**



Bininhalt: $N_i^t = \int_{x_i - b/2}^{x_i + b/2} f(x;a)dx$

folgt Poisson-Verteilung

Likelihood-Analyse $\mathcal{L}_i = \text{Poisson}(N_i; N_i^t) = \frac{{N_i^t}^{N_i}}{N_i!} \exp\left(-N_i^t\right)$

$$-\ln \mathcal{L}_i = -N_i \ln N_i^t + N_i^t + \cancel{\ln N_i!}$$ hängt nicht von der Funktion ab

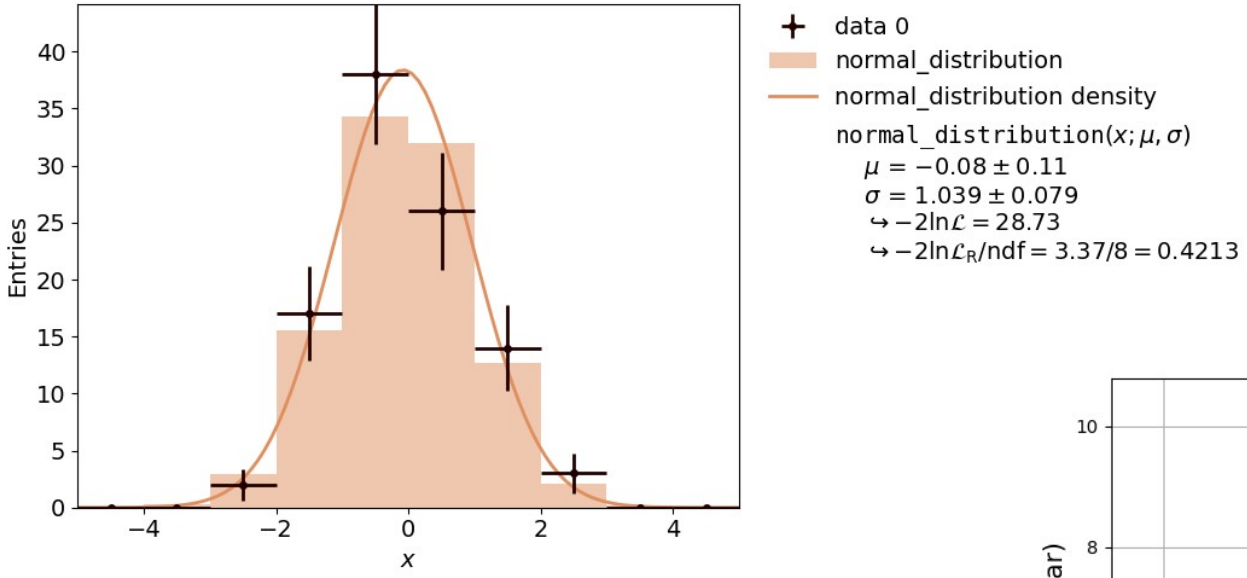
also: $F(\mathbf{a}) = -\sum_{i=1}^{N_{\text{bins}}} \ln \mathcal{L}_i(N_i; N_i^t(x_i; \mathbf{a}))$ minimieren bzgl. a

$\Rightarrow$ Schätzung für Parameter a

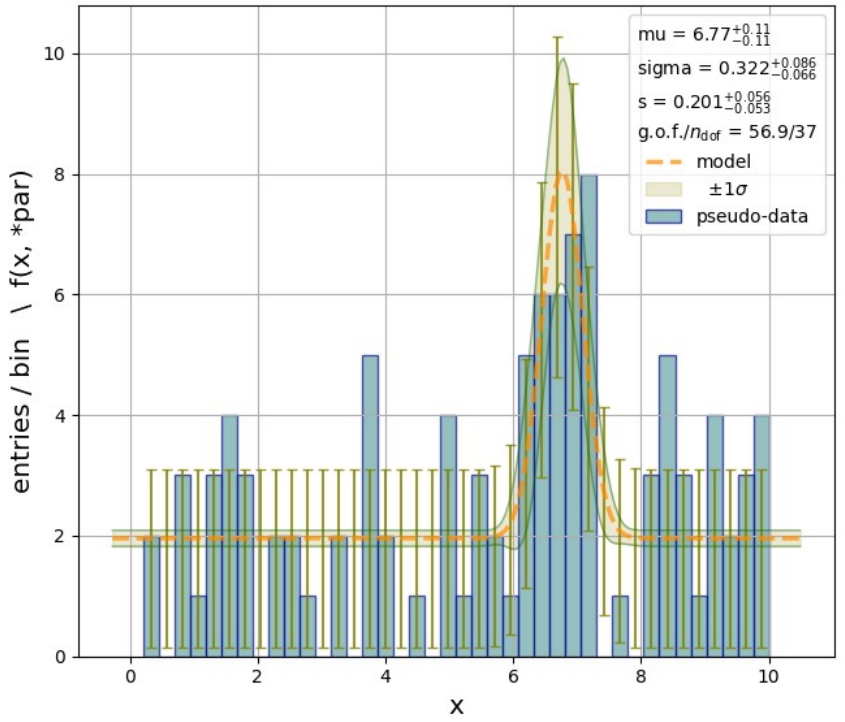
Implementiert in **PhyPrakit.phyFit.hFit** oder **kafe2**
sowie in **CERN Root**

Beispiele: **Binned In $\mathcal{L}$ - Fit**

kafe2/examples/009_histogram_fit/histogram_fit.py

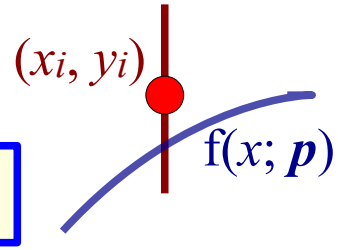


PhyPraKit//examples/test_hFit.py



Anpassung an (x, y) Daten

Betrachten **Messpunkte** $(x_i, y_i) = (\, x_i,\ f(x_i, \boldsymbol{p})\,)$



$$y_i = f(\, x_i,\ \mathbf{p}) + z_{y,i}$$

Unsicherheiten beschrieben durch korrelierte Zufallszahlen (z_i),
die einer multivariaten Gaußverteilung mit Kovarianzmatrix V folgen:

$$\mathcal{N}\,(\vec{z}, V)) = \frac{1}{\sqrt{(2\pi)^{n_d}\,\det(V)}} \cdot \exp\left(-\frac{1}{2}\vec{z}^{\,T} V^{-1} \vec{z}\right) .$$

Die **Likelihood** ist gegeben durch

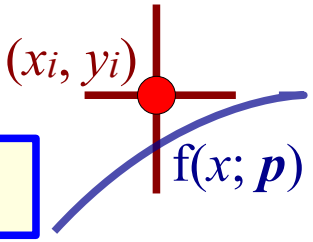
$$-2\,\ln \mathcal{L}_{\text{Gauss}}\left(\vec{y}, V, \vec{f}(\vec{x}, \vec{p})\right) = \left(\vec{y} - \vec{f}(\vec{x}; \vec{p})\right)^T V^{-1} \left(\vec{y} - \vec{f}(\vec{x}; \vec{p})\right) + \ln(\det(V)) + n_d\,\ln(2\pi)$$

$$= \chi^2\left(\vec{y}, V, \vec{f}(\vec{x}, \vec{p})\right) + \ln(\det(V)) + \text{const}\,.$$

Nur wenn die Kovarianzmatrix nicht von den Parametern
abhängt, reduziert sich der relevante Teil auf χ^2.

Im allgemeinen sind auch die
x - Werte mit Unsicherheiten behaftet:

$$y_i = f(\, x_i + z_{x,i}\,) + z_{y,i}$$



→

Behandlung von Unsicherheiten in x- und y-Richtung

1. Ableitung nutzen, um x-Fehler in y-Fehler umzurechnen und quadratisch zu den y-Fehlen addieren

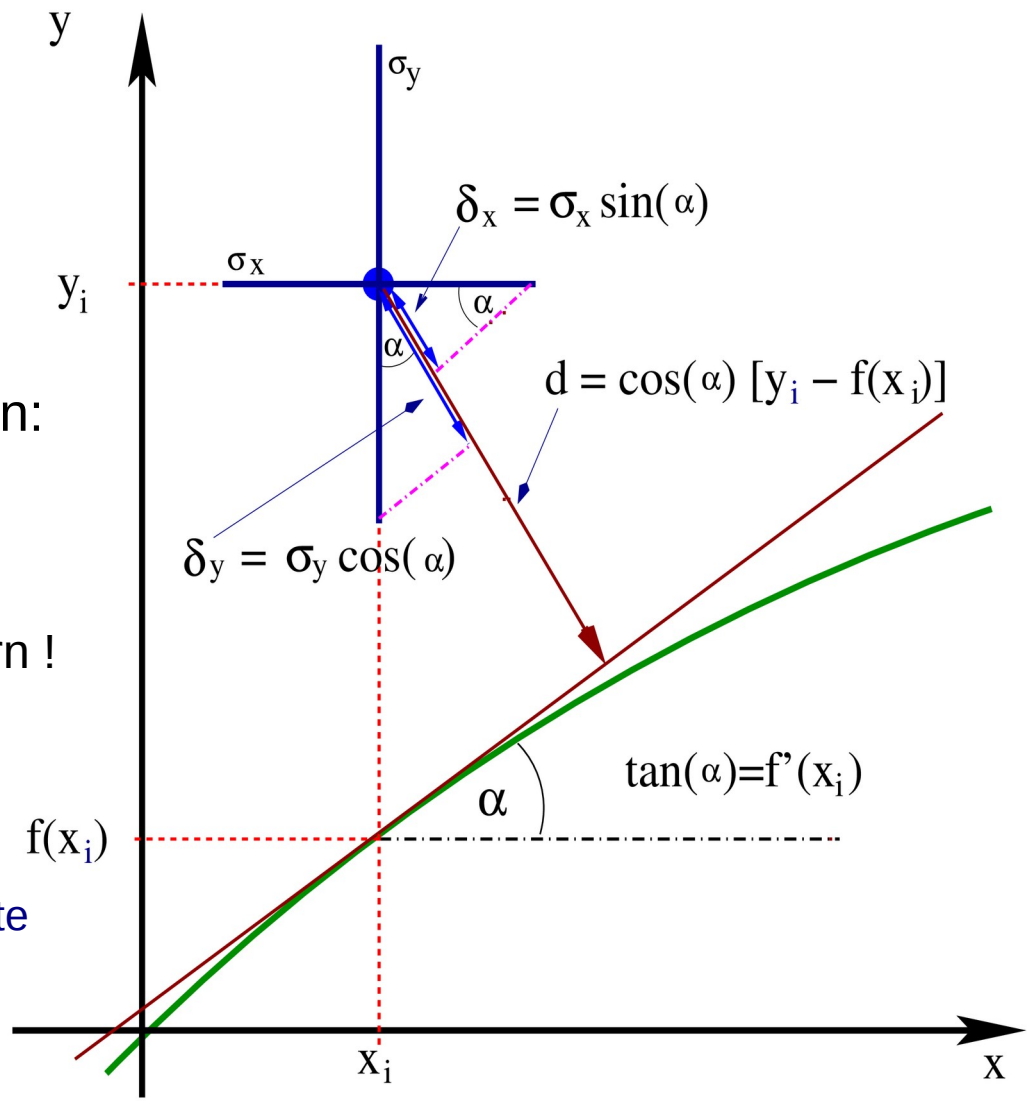
Iteratives Verfahren:

1. Anpassung ohne x-Fehler
2. $f'(x_i)$ bilden und neue Fehler bestimmen:

$$\sigma_i^2 = \sigma_{y_i}^2 + f'(x_i) \cdot \sigma_{x_i}^2$$

3. Schritt analog 2 zur Kontrolle wiederholen;

   χ^2 am Minimum darf sich nicht stark ändern !



Geometrische Interpretation:

Minimierung des auf projizierte Messfehler normierten Abstands d der Punkte von der Tangente an die Funktion:

$$\chi^2 = d^2 / (\cos(\alpha)\,\sigma_y^2 + \sin(\alpha)\,\sigma_x^2)$$

Implementierung: kafe2, PhyPraKit.phyFit, Root-Klasse TgraphErrors, scipy.odr

Behandlung von Unsicherheiten in x- und y-Richtung (2)

Allgemein bei korrelierten Unsicherheiten:

Kovarianzmatrizen V^x und V^y zur Gesamt-Kovarianzmatrix addieren:

$$V_{i,j} = V^y_{i,j} + V^x_{i,j} \cdot f'(x_i) \cdot f'(x_j)$$

Mit **genügend Rechenleistung** (heute kein Problem mehr) sollte die Kovarianzmatrix in jedem Schritt der numerischen Optimierung aktualisiert werden !

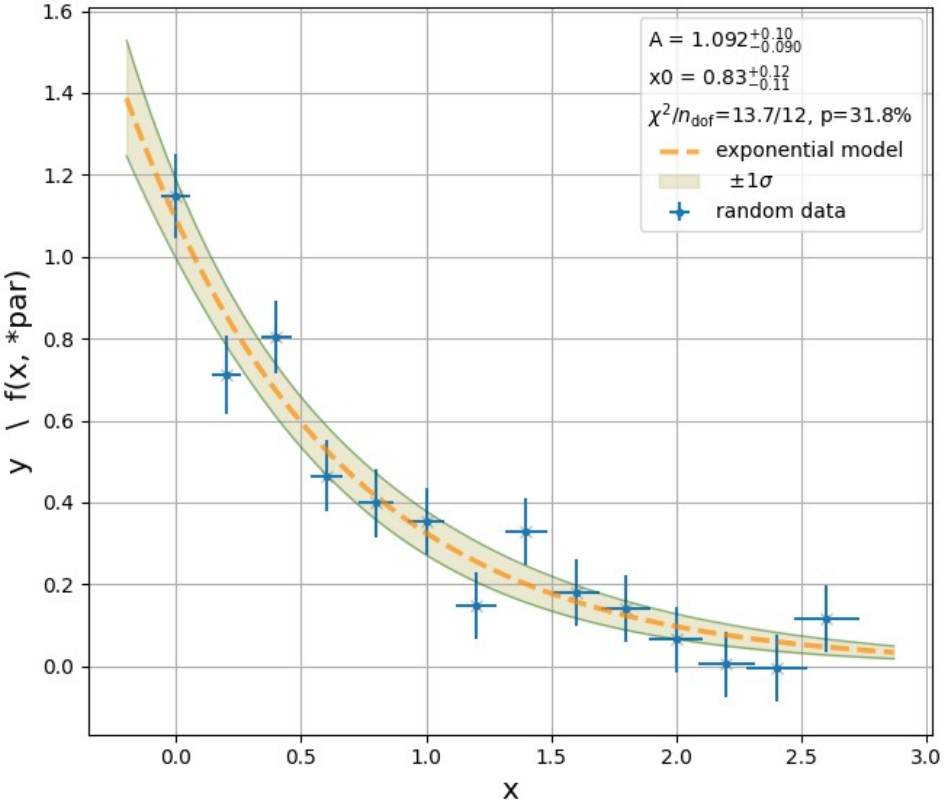
Achtung: Unsicherheiten sind parameterabhängig → nicht χ^2 sondern **Maximum Likelihood** verwenden !

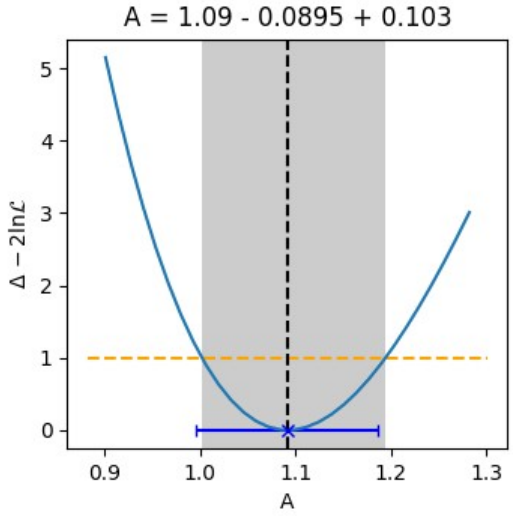
Implementiert in `kafe2, PhyPraKit.phyFit`

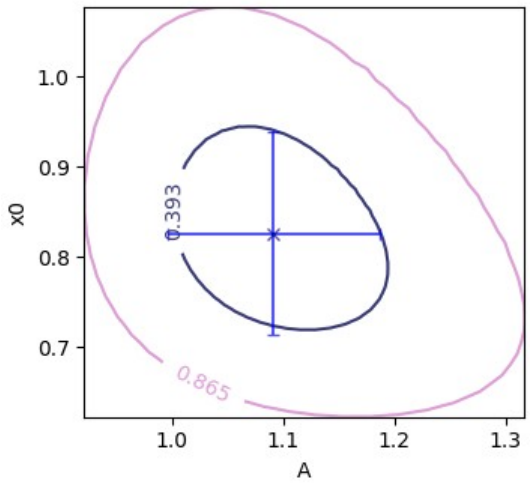
Beispiel: Modellanpassung an xy-Daten mit phyFit.xyFit()

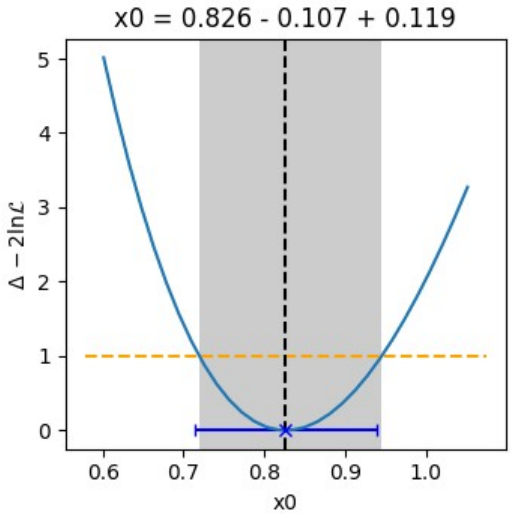
```
def model(x, A=1., x0=1.):
  return A*np.exp(-x/x0)
```

PhyPraKit//examples/test_xyFit.py









Likelihood mit mehreren Parametern:

Profile - Likelihood

Profile Likelihood

Oft interessieren nur wenige der Parameter einer Anpassung:

- $\boldsymbol{t} = (a_1, \ldots, a_k)$

die anderen sind „Störparameter“ (engl. nuisance parameters):

- $\boldsymbol{r} = (a_{k+1}, \ldots, a_m)$

$$p(\vec{x}\,|t, r) \rightarrow \mathcal{L}(t, r\,|\vec{x})$$

Frage: Unsicherheit von $\boldsymbol{t}$ für alle möglichen Werte von $\boldsymbol{r}$?

Antwort: Profile Likelihood $\mathcal{L}_{\text{prof}}(t) = \mathcal{L}(t, \hat{r}(t))$

Verfahren:

- zunächst Minimierung bzgl. $\boldsymbol{r}$ für verschiedene feste Werte von t_i
- 1σ-Intervall auf t bestimmt durch $\Delta\mathcal{L}_{\text{prof}}(t) = \frac{1}{2}$

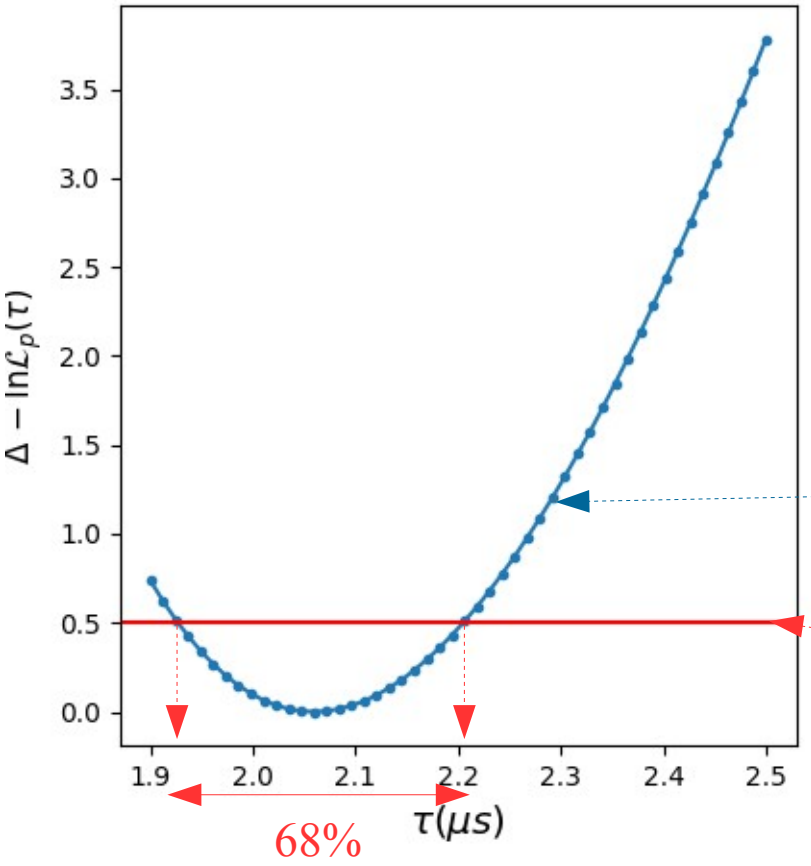
Bei mehreren Parmetern t_i : zur Fehlerbestimmung von t_i muss die Likelihood bzgl. aller anderen Parameter a_j; $j \neq i$ minimiert werden **(„Profile Likelihood“)**

(„MINOS“-Verfahren in MINUIT)

Beispiel zur Profile Likelihood

Das Verfahren in der Praxis am Beispiel der
Anpassung einer Exponentialfunktion an Zeitmessungen mit Untergrund
im Wertebereich $t_i \in [0, b]$:

$$p(t;\tau,f_b) = \frac{(1-f_b)}{\tau\,(1-\exp(-b/\tau))}\exp(-t/\tau) + \frac{f_b}{b}$$



von Interesse ist die Lebensdauer τ,
der Untergrundanteil f_b ist der Störparameter

Der Algorithmus:

- für feste Werte τ_i in der Nähe des Minimums wird $\mathcal{L}(\tau_i, f_b\,|\vec{t})$ bzgl. f_b minimiert

  $\rightarrow \quad \mathcal{L}_p(\tau_i\,|\vec{t})$

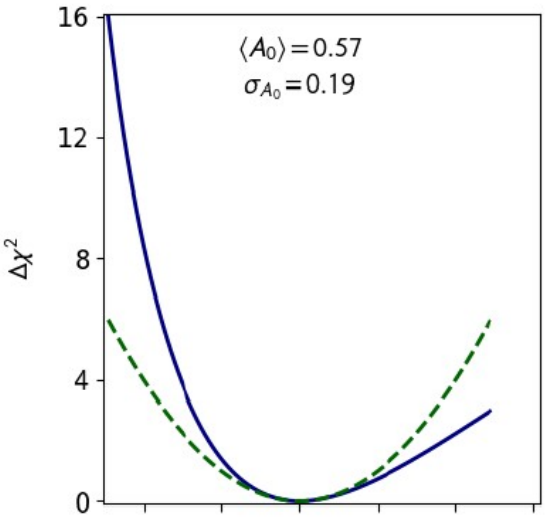
- das 68% CL - Intervall wird bei

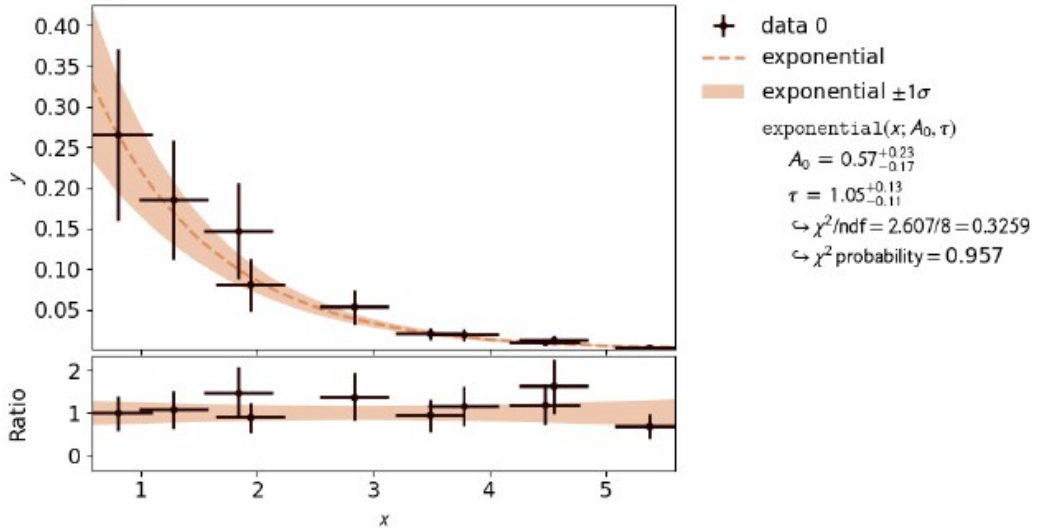
  $\mathcal{L}_p(\tau_{min}\,|\vec{t}) + \frac{1}{2}$ abgelesen

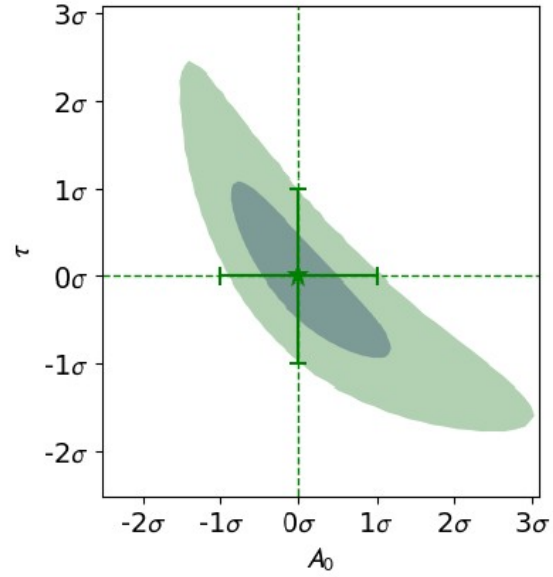
Skript: Übungsaufgabe !

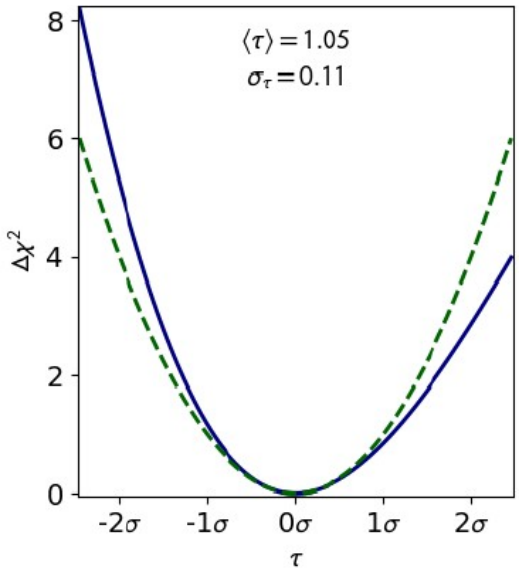
Beispiel 2 zur Profile-Likelihood

Profile-Likelihood in 2 Dimensionen zur Bestimmung von Konfidenzkonturen









- deutlicher Unterschied zwischen parabolischer Annahme und Profile-Likelihood
- Konfidenz-Konturen sind hier keine Ellipsen

erstellt mit kaf2/examples/003_profiling/01_non_linear_fit.py

Qualität der Anpassung

Goodness-of-Fit

Beurteilung der Güte einer Anpassung

Vergleich von Stichprobe aus Daten mit Modellen:
benötigen Maß für **Güte der Anpassung** (engl.: goodness of fit, GOF)

Goodness-of-Fit-Test = spezieller Hypothesentest (s. später):
Test der Nullhypothese (= Modell) ohne Spezifizierung alternativer Hypothesen

Modell verwerfen, falls schlechte Übereinstimmung !

Falls Wahrscheinlichkeitsverteilung der Stichprobe unbekannt:
→ standardisierte Goodness-of-Fit-Tests,
z. B. (der schon bekannte) $\chi 2$-Test

Erinnerung CgDA: Zusammenhang der Residuensumme S mit der χ^2 – Verteilung

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

Quadrat einer standard-normal verteilten Größe, folgt χ^2-Verteilung

Smin, **die „gewichtete Summe der Residuen-Quadrate“ am Minimum**,
folgt – bei Gauß-verteilten Fehlern σ_i – einer
χ^2-Verteilung mit $n_f = N\text{-}k$ Freiheitsgraden.
Erwartungswert: $\langle\chi^2\rangle = n_f$ oder $\langle\chi^2 / n_f\rangle = 1$

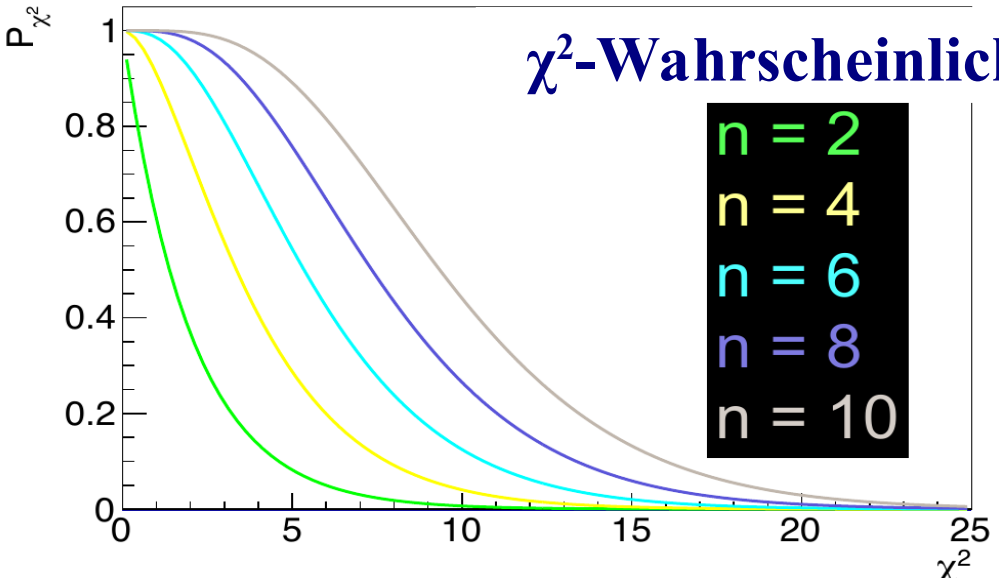
Die **χ^2-Wahrscheinlichkeit**

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

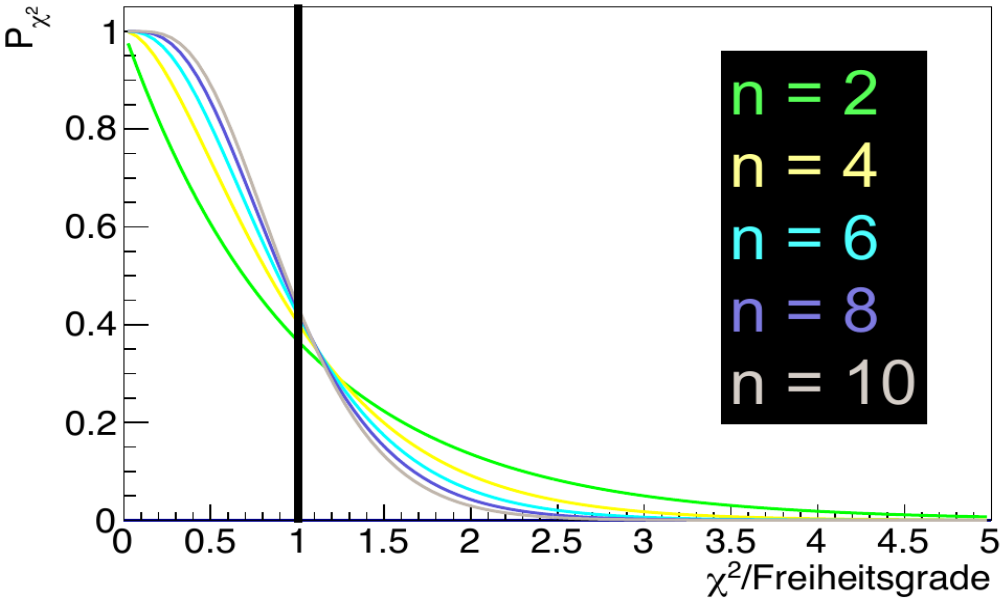
dient zur Quantifizierung der Qualität einer Anpassung

Aussage, mit welcher Wahrscheinlichkeit ein größerer Wert von χ^2 am Minimum als der tatsächlich beobachtete zu erwarten wäre.

Güte der Anpassung



χ^2-Wahrscheinlichkeit



in python:
`scipy.stat.chi2.cdf(S, nf)`

$$E[S_{min}] = nf$$

$$\rightarrow$$

$$E[S_{min} / nf] = 1$$

Die Wahrscheinlichkeit, einen χ^2 - Wert größer nf zu erhalten, ist ca. 40%

(ziemlich unabhängig von n)

Mit wachsender Zahl der Freiheitsgrade sollte S_{min} / nf immer näher bei 1 liegen

in python: `chi2prb= 1. - scipy.stat.chi2.cdf(S, nf)`

Qualität der Anpassung aus Likelihood

Likelihood-Methode zur Parameteranpassung liefert zunächst
keine Information zur Qualität der Übereinstimmung von Daten und Modell

Durch geeignete „Normierung“ kann Qualitätsinformation auch aus der Likelihood gewonnen werden: **Likelihood-Verhältnis**

der beobachteten Daten und (geeigneter) Referenz.

Referenzmodell z.B. ein fiktives „Fully Saturated Model“,
das die Daten exakt beschreibt, $m_{sat}(x_i) = y_i$

Beispiel: Likelihood der Gaußverteilung

$$-\ln\mathcal{L}(\vec{a}|\vec{y}) = \frac{1}{2}\sum_i \left(\frac{y_i - f(x_i,\vec{a})}{\sigma_i}\right)^2 + \sum_i \ln(\sqrt{2\pi}\sigma_i)$$

$$-\ln\mathcal{L}_{\rm sat}(\vec{a}|\vec{y}) = \frac{1}{2}\sum_i \left(\frac{y_i - m_{\rm sat}(x_i)}{\sigma_i}\right)^2 + \sum_i \ln(\sqrt{2\pi}\sigma_i) = 0 + \sum_i \ln(\sqrt{2\pi}\sigma_i)$$

$$\Rightarrow \text{-ln}\left(\frac{\mathcal{L}(\vec{a})}{\mathcal{L}_{\rm ref}(\vec{a})}\right) = \sum_i \underbrace{\left(\frac{y_i - f(x_i,\vec{a})}{\sigma_i}\right)^2}_{\chi^2} + \underbrace{\sum_i \ln(\sqrt{2\pi}\sigma_i) - \left(0 + \sum_i \ln(\sqrt{2\pi}\sigma_i)\right)}_{0} = \chi^2(\vec{a})$$

in früheren Diskussionen hatten wir diesen Term auf andere Art zu Null diskutiert !

(„hängt ja gar nicht von den Parametern ab -kann man also weglassen!“)

Qualität der Anpassung: binned Likelihood-Fit

Beispiel 2: Likelihood der Poisson-Verteilung $\mathrm{Poiss}(n_i;\mu_i) = \frac{{\mu_i}^{n_i}}{n_i\,!}\,\mathrm{e}^{-\mu_i}$

$$-\ln\mathcal{L}_i = \sum_i -N_i \ln\mu_i + \mu_i + \ln N_i\,!$$

Referenz: $\mu_i = N_i,\ -\ln\mathcal{L}_{\mathrm{sat}} = \sum_i -N_i\ln N_i + N_i + \ln N_i!$

$$\Rightarrow \quad \ln\frac{\mathcal{L}(\vec{\hat{a}})}{\mathcal{L}_{\mathrm{ref}}(\vec{\hat{a}})} = \sum_i N_i \ln\frac{N_i}{\mu_i} + \mu_i - N_i =: \mathrm{gof}_{\mathrm{Poisson}}$$

gof = goodness of fit

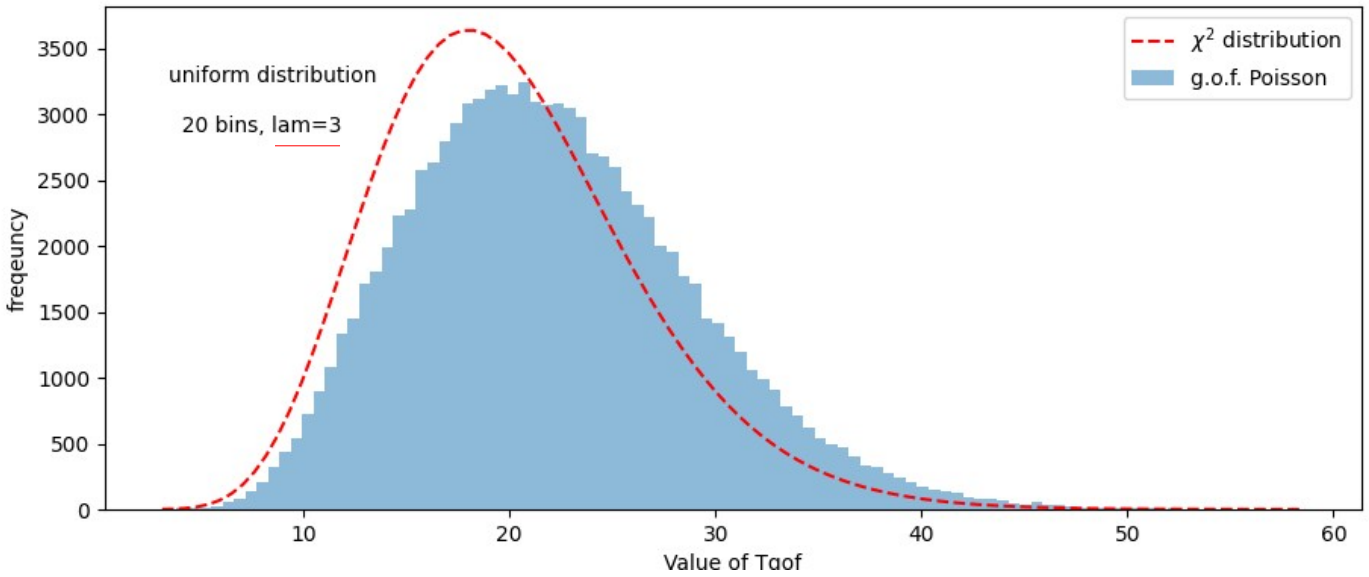
$\mathrm{gof}_{\mathrm{Poisson}}$ konvergiert für große c_i gegen $\chi^2/2$

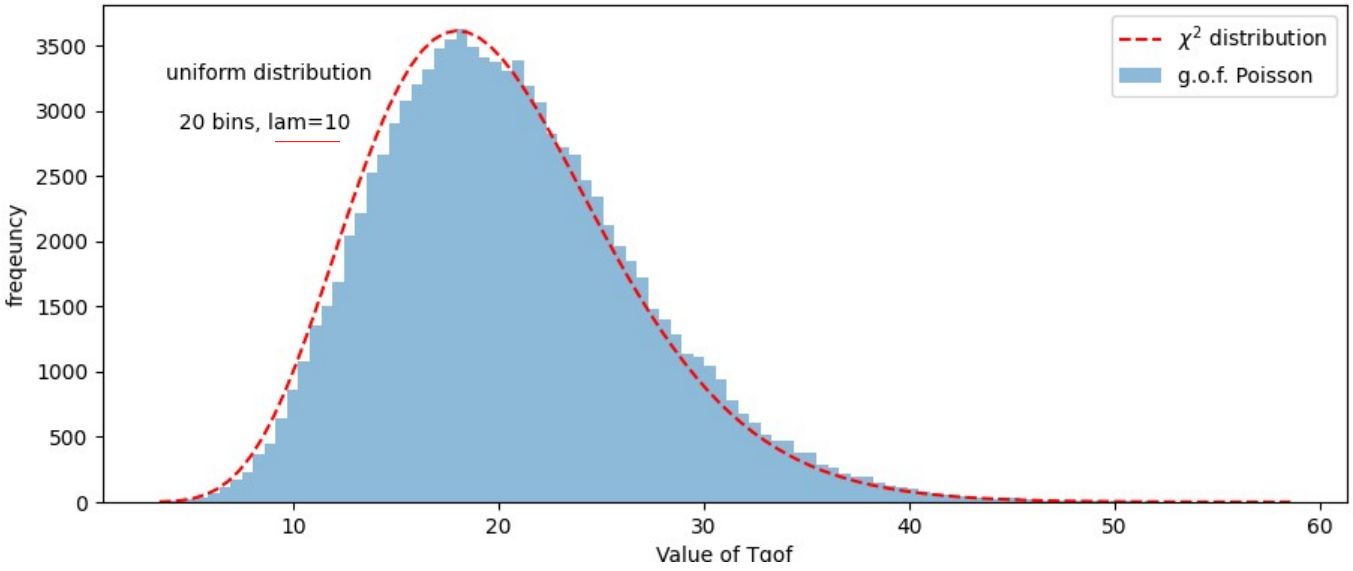
Script gof-test.py

Auf analoge Weise lassen sich (effiziente) **Prüfgrößen** auf Basis des Likelihood-Verhältnisses auch für andere Verteilungen generieren

Qualität der Anpassung: binned Likelihood-Fit (2)

$gof_{Poisson}$ nähert sich für große“Bin-Einträge der χ^2 – Verteilung an

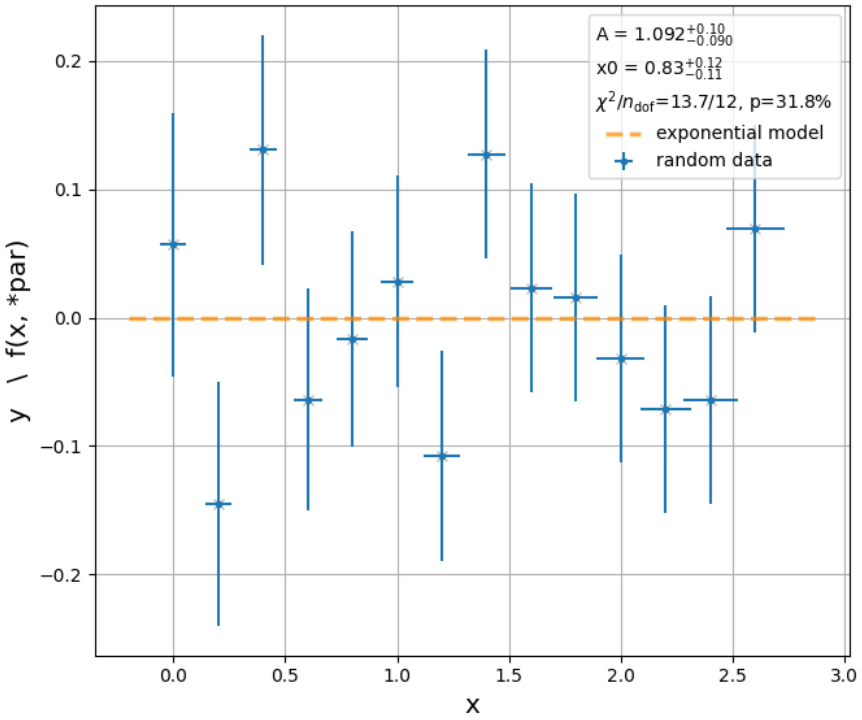




Übereinstimmung Daten mit Modell: „Pull“

Die auf die Unsicherheit normiert Abweichung der Daten vom Modell nennt man $\mathrm{pull}_i = \dfrac{y_i - f(x_i, \hat{\mathbf{p}})}{\sigma_i}$

Beispiel von eben: xy-Fit mit Residuen



Diese Form der Darstellung erlaubt es, Abweichungen der Datenpunkte vom Modell klar zu identifizieren.

Im Idealfall sind die **Pulls standard-normal verteilt** !

Ensemble-Tests mit Monte Carlo-Methode

Bei **Parameterschätzungen aus kleinen Stichproben** oder zur Überprüfung der Eigenschaften von Anpassungsverfahren bleibt oft nur die wiederholte Anwendung auf einer großen Anzahl simulierter Stichproben.

Lösung mit Monte Carlo-Methode:

Pseudoexperimente (weitere Bezeichnungen: Ensembletest, Toy-MC)

- Modell f(x;a) als bekannt vorausgesetzt → generiere viele. ggf. stark vereinfachte Stichproben („Pseudoexperimente") basierend auf f(x;a)
- Analysiere jedes Pseudoexperiment wie durchgeführtes Experiment
- Extrahiere statistische Eigenschaften aus Ensemble der Resultate

⇒
- Information über mögliche Verzerrungen $\text{Verteilung von } (\hat{p} - p_0)$
- Bestimmung der Coverage $W(p_i \in [\hat{p}_i - \Delta p_i, \hat{p}_i + \Delta p_i]) = 68.3\%$
- Korrelationen der Parameter $\text{Streudiagramme der Parameter}$

Beispiel: Ensemble-Test

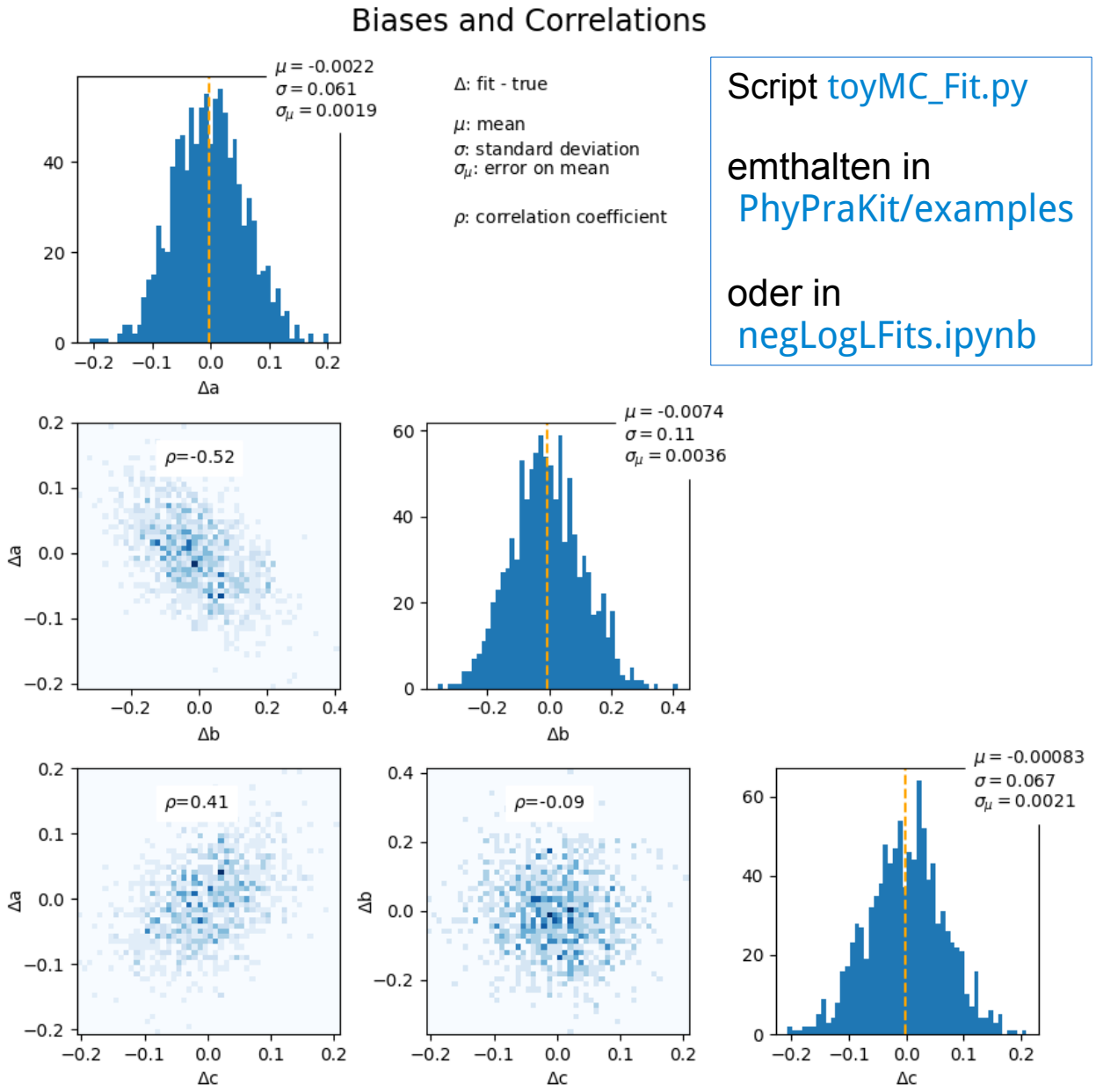
Beispiel:

Anpassung einer
Parabel an Daten
mit Unsicherheiten
in x- und y-Richtung

(1000 Pseudoexeperimente)

Ausgabe:

```
* biases:
  0: -0.00235 +\- 0.0019,  std 0.0605
  1: -0.00754 +\- 0.0036,  std 0.114
  2: -0.000882 +\- 0.0021,  std 0.0667
* coverage:
  0: 97.4%
  1: 96.8%
  2: 98%
```



Script toyMC_Fit.py

emthalten in
PhyPraKit/examples

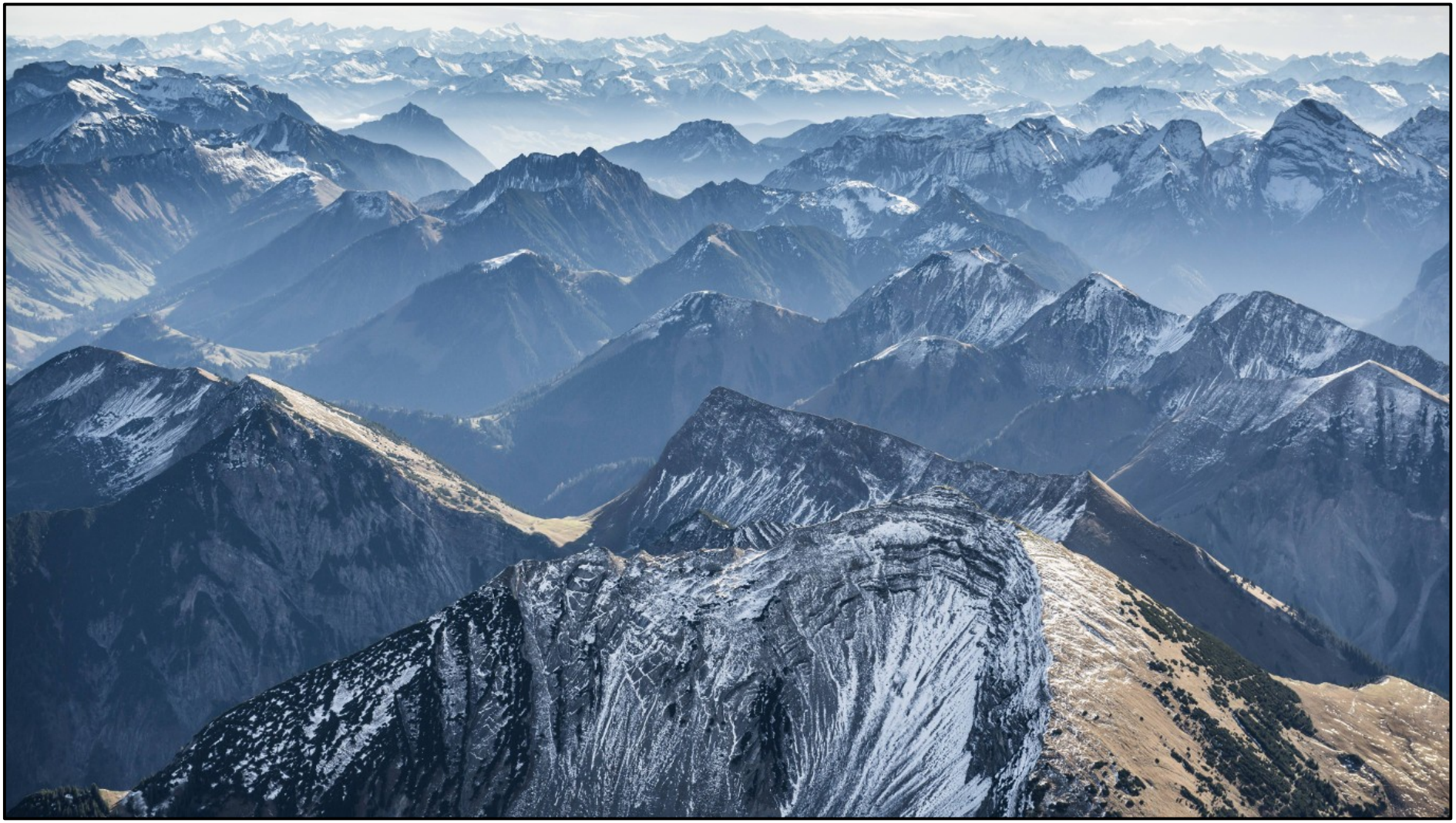
oder in
negLogLFits.ipynb

Numerische Verfahren
zur Minimierung v. Kostenfunktionen

Die Problemstellung:

Suche nach dem Minimum einer skalaren Funktion in n_p Dimensionen

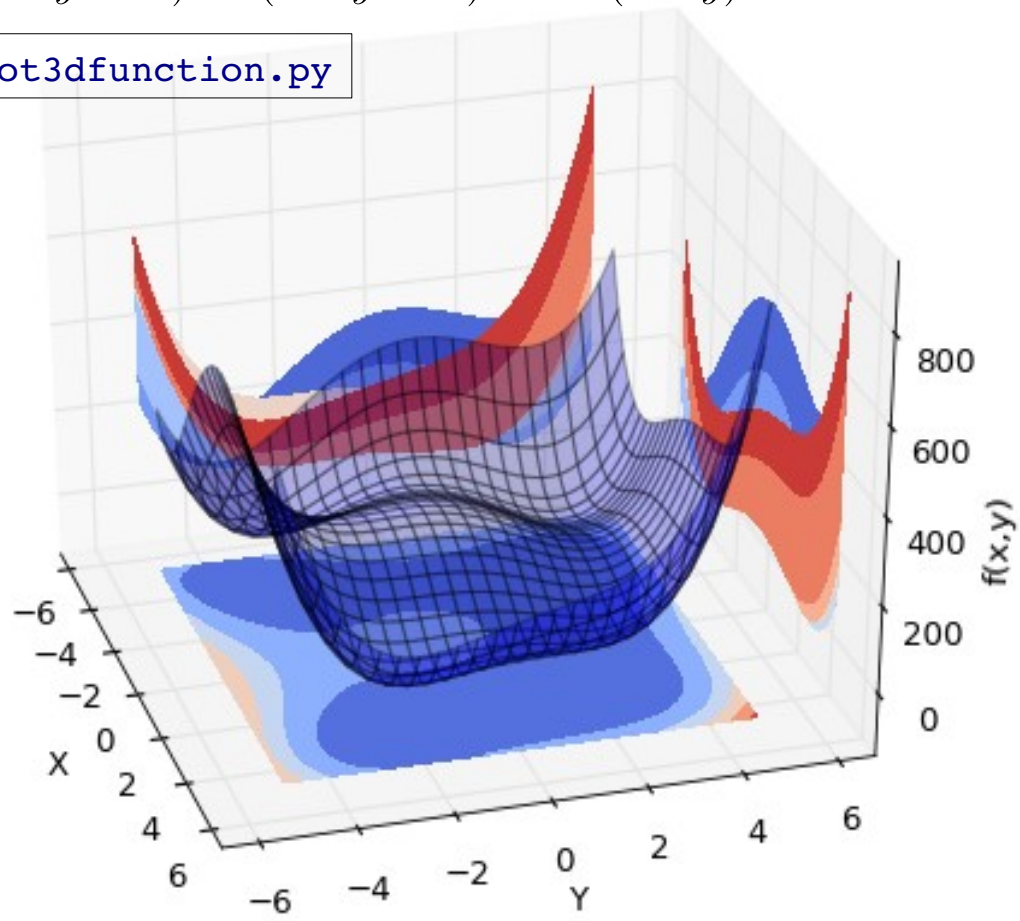
äquivalent: Wo ist das tiefste Tal ?



Numerische Optimierung (=Minimierung)

$$f(x,y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2 + 0.1(x + y)^2$$

s. Beisp. `plot3dfunction.py`



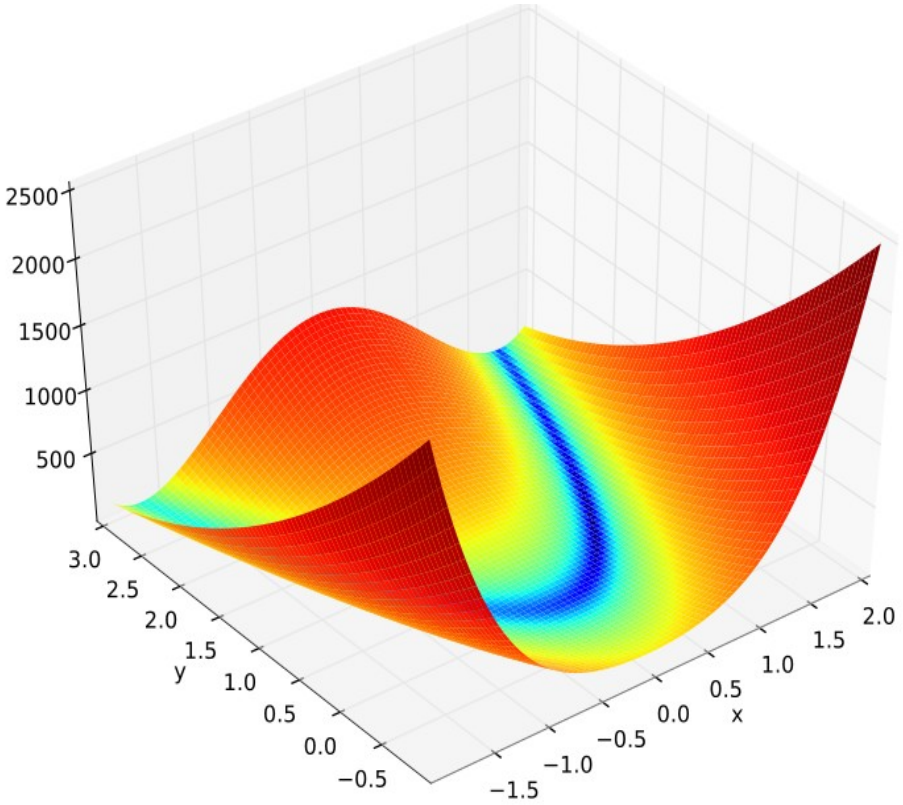
Minimierungsalgorithmen konvergieren i.a. gegen lokale Minima

Es gibt keine allgemeine Methode zum Finden des globalen Minimums

x* ist ein lokales Minimum, wenn

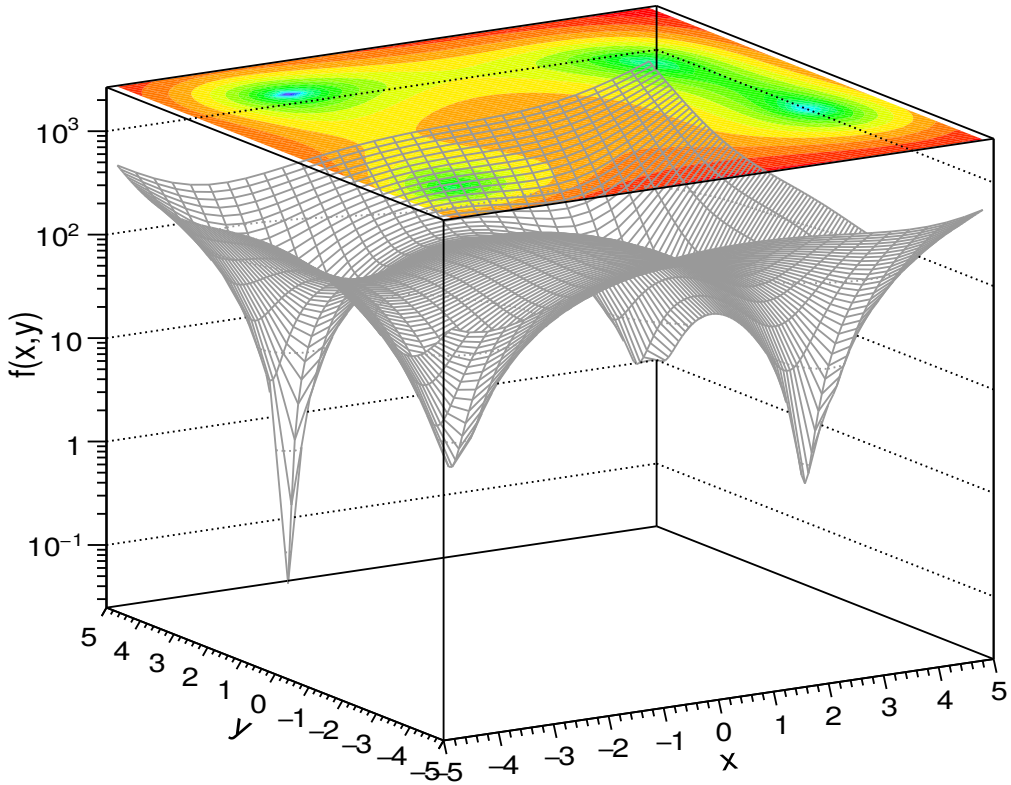
$F(x^*) < F(x)$ für alle $x \neq x^*$ in der Umgebung von x^*

Achtung: Auffinden des globalen Minimums ist nicht trivial !



Die Rosenbrock-Funktion:

ein langes Tal mit flachem Minimum



Modifizierte Rosenbrock-Funktion:

vier Minima unterschiedlicher Tiefe

(im log-Plot sehr deutlich zu sehen)

Häufig genutzte Benchmark-Funktionen für Minimierer !

Numerische Anpassung

I. a. (heute) Anpassungen mittels **numerischer Optimierung** durch:

Gittermethode:

z.B. Minuit: SCAN-Algorithmus

- k gleichverteilte Testwerte pro Dimension
  - → Erfordert k^d Berechnungen bei d Dimensionen
    Ungeeignet für große d

Monte-Carlo-Methode:

- Funktionsberechnung an zufällig verteilten Testpunkten
  - → Auch bei großen *d* geeignet
    gut für Bestimmung von Startwerten

Einfache Parametervariation:

- Eindimensionale Minimierung in einem Parameter, dann Minimierung in nächstem Parameter → Iteration
  - → **I.a. schnelle Konvergenz, wenn Minimierung in Richtung der Eigenvektoren der Matrix der 2. Ableitung erfolgt**

weitere numerische (deterministische) Verfahren:

- Simplex-Verfahren (ohne Ableitungen)

z.B. Minuit: SIMPLEX-Algorithmus

- Methoden mit angepasster Schrittweite, numerische Auswertung der Ableitungen

MIGRAD-Algorithmus

Beispiel **Simplex-Verfahren**

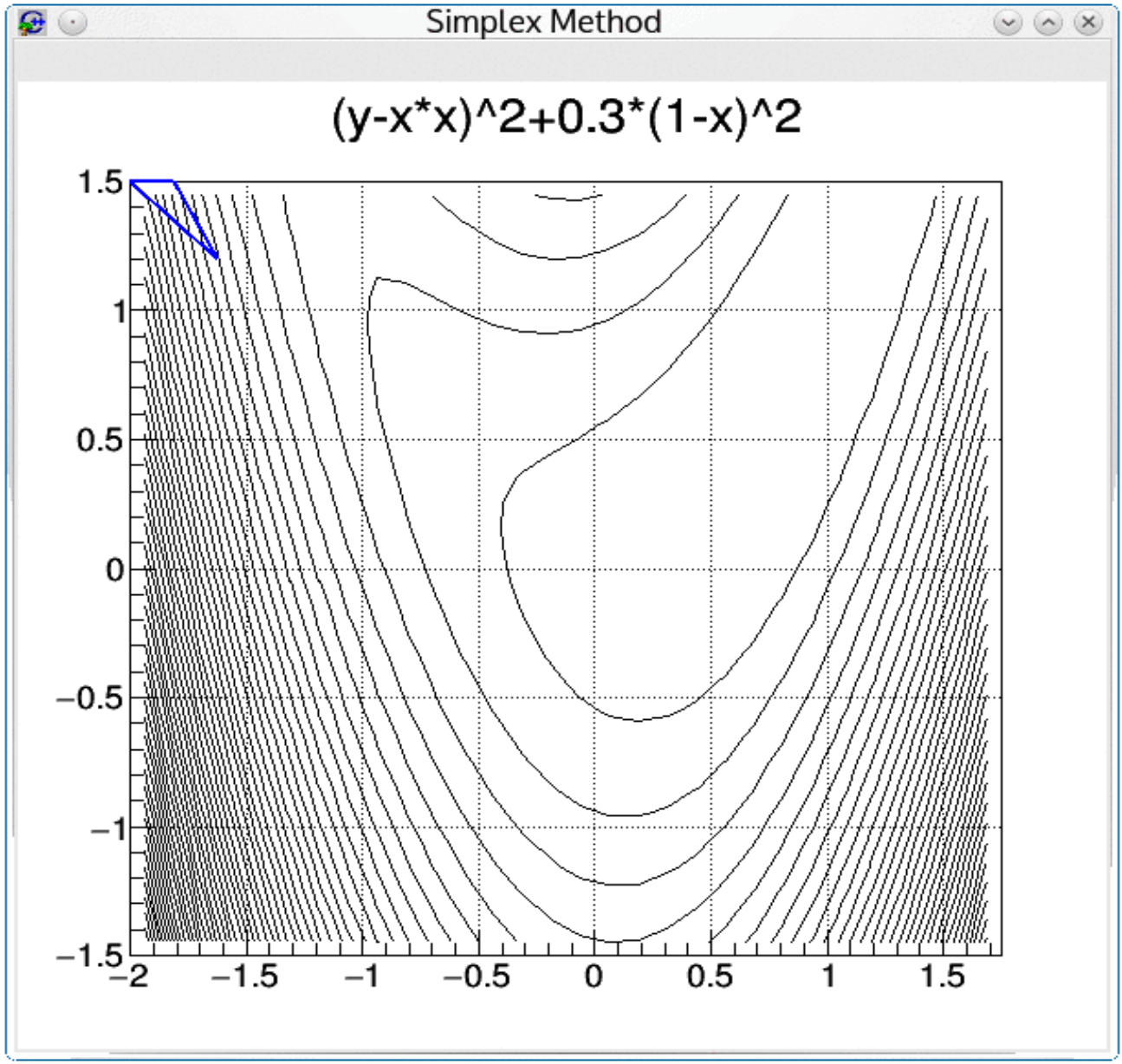
Illustration der **Simplex-Methode**

am Beispiel der
Rosenbrock'schen
„Bananenfunktion“

Simplex:

n-dimensionaler Polyeder aus *n+1* Punkten im R^n

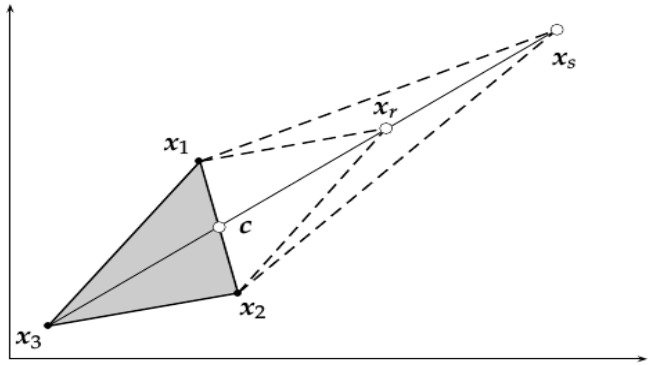
Bei jedem Schritt wird der schlechteste Punkt $\mathbf{x}_r$ mit größtem Funktionswert $F(\mathbf{x}_r)$ durch einen neuen, besseren, ersetzt, oder der Simplex ggf. verkleinert oder vergrößert.



Simplex-Verfahren

n+1 Punkte x_1 , ..., x_{n+1} im R^n

n-dimensionaler Polyeder oder „Simplex“



- Sortierung, so dass $F(x_1) < \ldots < F(x_{n+1})$
- Schwerpunkt der n besten Punkte: $c = \sum_{i=1 \ldots n} x_i / n$
- Spiegelung des schlechtesten Punktes an c: $x_r = c + \alpha\,(c - x_{n+1})$

  Falls $F(x_1) < F(x_r) < F(x_n)$ → x_r ersetzt x_{n+1}

  Falls $F(x_r) < F(x_1)$ *gute Richtung* → Streckung: $x_s = c + \beta(x_r - c)$, $\beta > 1$

  Falls $F(x_s) < F(x_r)$: x_s ersetzt x_{n+1} , ansonsten x_r ersetzt x_{n+1}

  Falls $F(x_r) > F(x_n)$: *Simplex zu groß*

  → Abflachung: $x_s = c - \gamma\,(c - x_{n+1})$, $0 < \gamma < 1$

  Falls $F(x_s) < F(x_{n+1})$: x_s ersetzt x_{n+1}
- Ansonsten Kontraktion um x_1 : $x_j = x_1 + \delta\,(x_j - x_1)$, $0 < \delta < 1$

Gradientenabstiegsverfahren

Grundidee aller **Abstiegsverfahren**:

- Suche Abstiegsrichtung $\boldsymbol{d}_i$ der Verlustfunktion $f(\boldsymbol{x})$ an Punkt $\mathbf{x}_i$
- Nächster Punkt: $\mathbf{x}_{i+1} = \mathbf{x}_i + t_i\, \mathbf{d}_i$ (mit Schrittweite t_i)
- Wiederhole bis Abbruchkriterium erfüllt ist

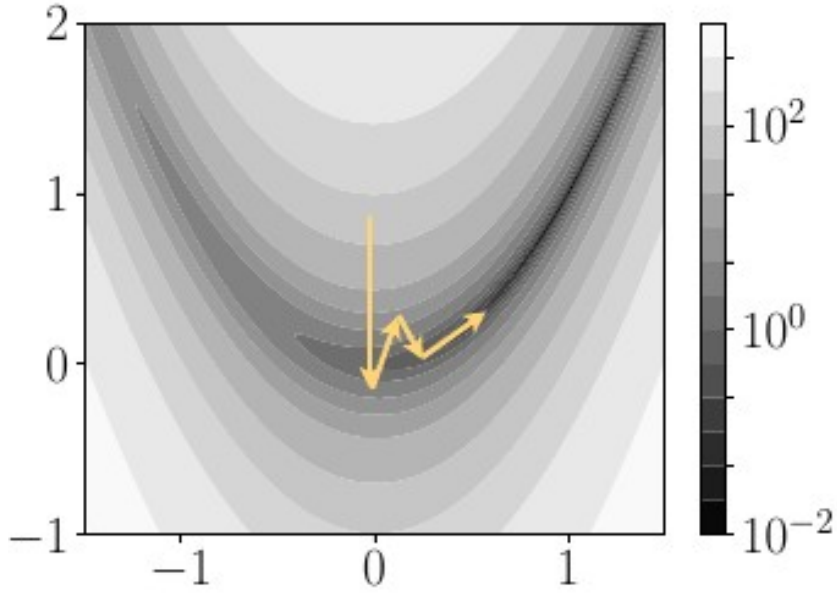
Algorithmus 1. Ordnung: **Gradientenverfahren**

(engl.: gradient descent, steepest descent)

- Suche Richtung des steilsten Abstiegs über Gradient ∇f, benötigt 1. Ableitung:

$$\mathbf{d}_i = \nabla f(\mathbf{x}_i)$$

- t_i bestimmt über Liniensuche so dass f($\mathbf{x}_{i+1}$) möglichst klein wird



Gradientenverfahren (2)

Verbesserungen des **Gradientenverfahrens**:

- **Konjugierte Gradienten** (engl.: conjugate gradients, CG):
  schnellere Konvergenz durch Schritte in „konjugierte" Richtungen
  (exakt für Minimierung quadratischer Formen, sonst iterativ)
- **Stochastisches Gradientenverfahren** (engl.: stochastic gradient descent)
- **Adaptive Schrittweite**, ggf. unterschiedlich für jeden Parameter
- **Impuls:** physikalische Analogie – zu minimierende Funktion = Potenzial,
  das Kraft auf Probeteilchen ausübt
  → Impuls proportional zu Gradient des Potenzials
  → frühere Gradienten erzeugen Trägheit in Abstiegsrichtung

Aktuelles Beispiel: Adam (https://arxiv.org/abs/1412.6980)

Abstiegsverfahren: Newtwon-Verf.

Algorithmus zweiter Ordnung: Newton-Verfahren

Erinnerung: iteratives Verfahren zur Bestimmung von Nullstellen (in 1D) f (x)

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

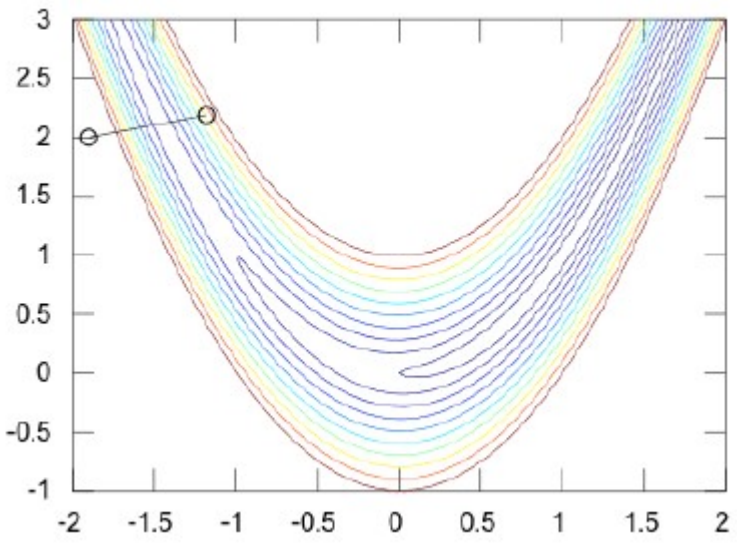
- **Hier:**

  suche Nullstelle von ∇f über **Hesse-Matrix** $\nabla^2 f$ *(benötigt 2. Ableitung)*

$$\mathbf{d}_i = -(\nabla^2 f(\mathrm{x_i}))^{-1}\, \nabla f(\mathrm{x}_i)$$

- Verfahren mit **variabler Metrik**
  (auch: Quasi-Newton-Verfahren):
  aufwändige Berechnung der Hesse-Matrix vermeiden → **iterative Näherung**
  (z. B. BFGS = Broyden, Fletcher, Goldfarb, Shanno; DFP = Davidon, Fletcher, Powell)

Riedel, Rosenbrock-bfgs-animation.gif, gemeinfrei



„Simuiertes Ausglühen“ – Simulated Annealing

Monte Carlo-Methoden sind sehr effizient bei der Suche über große Bereiche in hochdimensionalen Problemen –

sehr gut geeignet zur Bestimmung von Startwerten

Beispiel: Simulated Annealing *(Übungsaufgabe)*

Idee aus der Physik entlehnt:

Abkühlung eines thermodynamischen Systems, das der Boltzmann-Statistik genügt

Wahrscheinlichkeit, Mikrozustand E_j anzutreffen: $p(E_j) \propto \exp(-\frac{Ej}{k_B T})$

energetisch günstigster Zustand E_0 wird durch Abkühlen, T→0, erreicht.

Anschaulich: „Herausschütteln“ aus lokalen Minima wie Reis in einem Sieb.

Die Energie aus dem physikalischen Beispiel entspricht allgemein der „Kostenfunktion“ *F*

also –ln $\mathcal{L}$ oder χ^2 im Falle der Parameteranpassung

transformiere $F(x) \rightarrow \tilde{F}(x) = \exp\left(-\frac{F(x)}{T}\right)$ („Boltzmann-Faktor“);

für zwei Zustände i, i+1 gilt $\frac{\tilde{F}(x_i)}{\tilde{F}(x_{i+1})} = \frac{\tilde{F}_i}{\tilde{F_{i+1}}} = \exp\left(-\frac{F_{i+1} - F_i}{T}\right)$

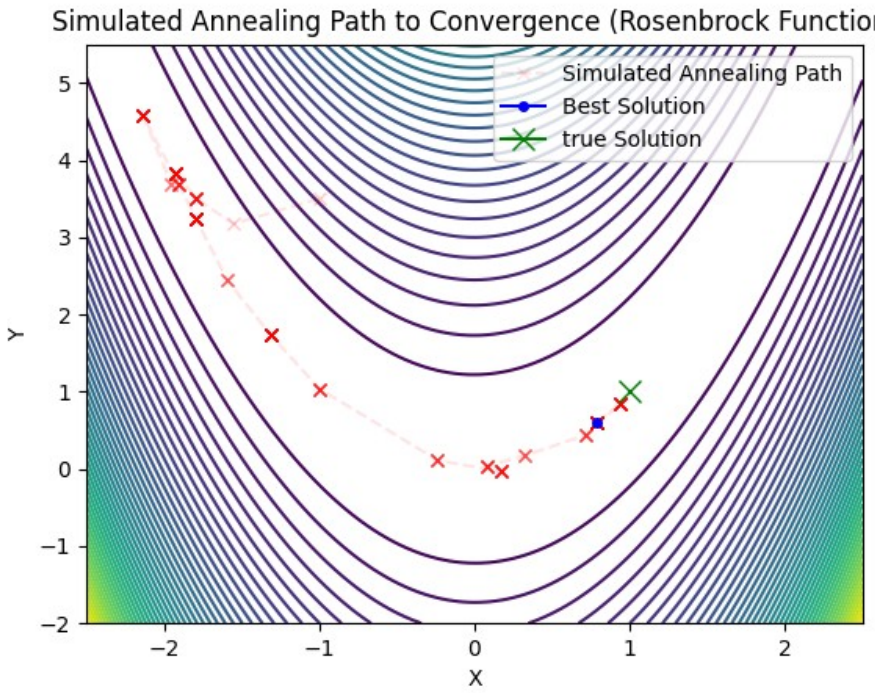
Simulated Annealing – Diskussion des Verfahrens

Simulated Annealing:

- Beispiel für mathematischen Algorithmus mit physikalischer Analogie
- Vergleich mit Boltzmann-Faktor:

  → endliche Wahrscheinlichkeit, aus Nebenminima herauszuspringen
- Abkühlung: Wahrscheinlichkeit sinkt im Laufe der Optimierung
- Beispiel für Markov-Chain Monte-Carlo (MCMC, N. Metropolis, 1953)

Skript: simulAnneal_chatGPT.py



Minimierung der „Rosenbrock-Funktion“ mittels Simulated Annealing

Simulated Annealing – der Algorithmus

Der Algorithmus:

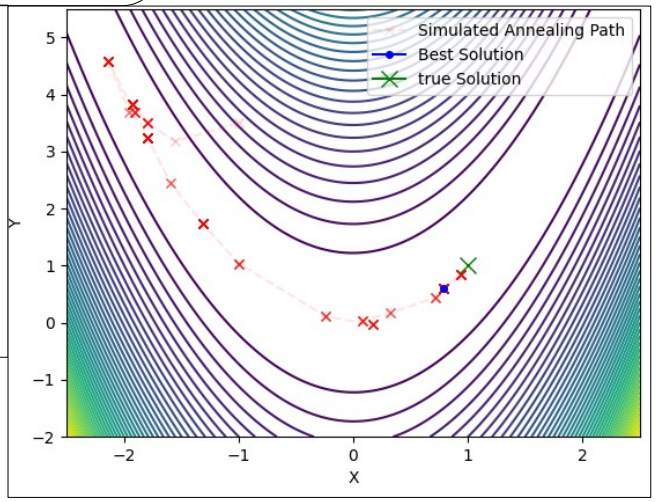
1. Start mit einem zufälligen Zustand x_1 bei der Temperatur T
2. erzeuge zufälligen neuen Zustand x_{n+1}
3. akzeptiere x_{n+1}, wenn $F_{n+1} < F_n$

   oder wenn $r < \exp\left(-\frac{F_{n+1}-F_n}{T}\right)$ mit Zufallszahl $r \in]0,1]$
4. wiederhole Schritt 2 bis 3 k-mal
5. kontrolliere Konvergenz
6. reduziere Temperatur um ΔT und gehe zu Schritt 2, falls Endtemperatur noch nicht erreicht ist

Zahlreiche freie Parameter:

- Anfangstemperatur ***T***
- Zahl ***k*** der Iterationen bei gleicher Temperatur
- das „Kühl-Schema“, also die Bestimmung von ***ΔT***
  (linear, exponentiell, logarithmisch, oder „adaptiv“)

Erfolgreiches „annealing“ benötigt Fein-Abstimmung und Kontrolle



Numerische Optimierung: Werkzeuge

- scipy.optimize: Sammlung von Python-Funktionen zur Optimierung
  - minimize(): diverse Algorithmen, voreingestellt: Nelder-Mead (Simplex)
  - curve_fit(): nichtlineare kleinste Fehlerquadrate zur (einfachen) Funktionsanpassung

- MINUIT: CERN-Programmbibliothek zur Funktionsminimierung
  (FORTRAN → C++)

  Aktuell: Minuit2 (integriert in ROOT) mit Python-Anbindung (iminuit)

  Algorithmen: MIGRAD (variable Metrik, BFGS/DFP), SIMPLEX, MINOS (Profile-Likelihood und Kontur-Scan, asymmetrische Parameterunsicherheiten)

Wie immer:
das Kleingedruckte lesen → benutzen Sie gut getestete Programme !

Werkzeuge zur numerischen Anpassung von Modellen

Software-Werkzeuge zur Parameteranpassung

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
- 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 auch andere gebräuchlich)
- Ausgabe der Korrelationen bzw. Kovarianz-Matrix der Parameter
- graphische Darstellung der Modellunsicherheit
- asymmetrische Fehler mittels Scan der (Profile-) Likelihood - Kurve am Minimum
- Ausgabe von Kovarianz-Ellipsen (d. h. Wahrscheinlichkeitskonturen)

kafe2 und PhyPraKit.phyFit unterstützen alle der genannten Punkte;
fast alle großen Projekt in der Experimentalphysik haben eigene, spezialisierte Pakete.

Vorbereitung VL06: Hypothesentest

- Testen von Hypothesen V06a_Hypohesentest