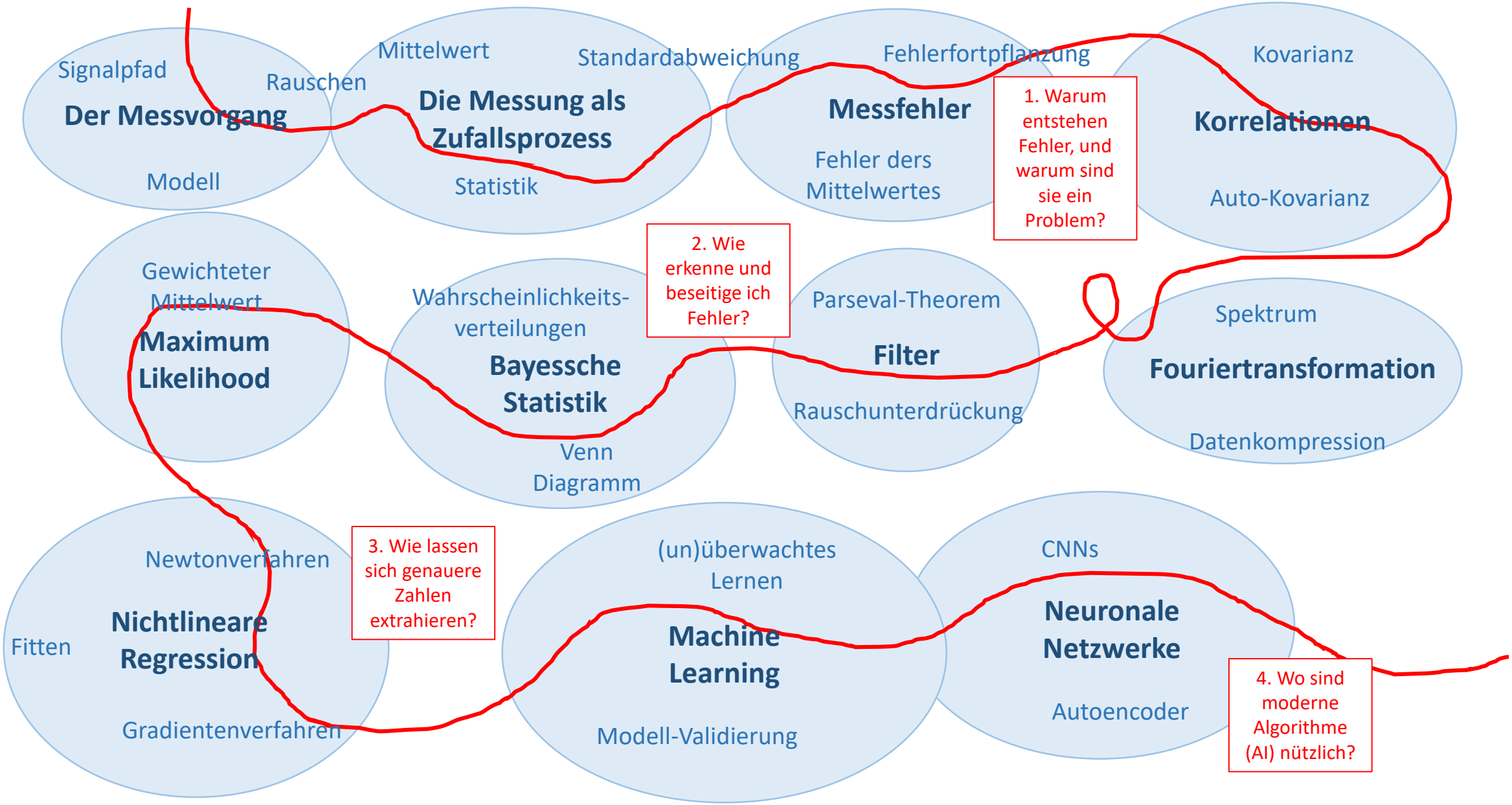
Fitten von nichtlinearen Funktionen

Der rote Faden: wie erhalte ich aussagekräftige Zahlen aus komplexen Daten?



Themen dieser Vorlesung:

- Nichlineare schätzung der kleinsten Quadrate
- Gradientenverfahren
- Newtonverfahren
- Marquardts Methode
- Berechnen der Kovarianzmatrix aus der Hesse Matrix

Repetition

1. Was ist eine Likelihood Funktion

2. Welche Klasse von Funktionen können wir mit der linearen Regression an einen Datensatz fitten?

Repetition

1. Was ist eine Likelihood Funktion

- Ist eine Plausibilitätsfunktion.
- Wird aus einer Wahrscheinlichkeitsverteilung gewonnen.
- Ist eine Funktion der Parameter die die zugrunde liegende Wahrscheinlichkeitsfunktion beschreiben.
- Die Likelihood Funktion ist maximal für die plausibelsten Werte der Parameter die die zugrunde liegende Wahrscheinlichkeitsfunktion beschreiben.

2. Welche Klasse von Funktionen können wir mit der linearen Regression an einen Datensatz fitten?

Mit der Methode der linearen Regression erhalten wir eine analytische Lösung für das Polynom das die Daten am besten wiedergibt.

Nichlineare schätzung der kleinsten Quadrate

Erinnerung:

Um die lineare Funktion $a_0 - a_1 x_n$ an einen gemessenen Datensatz $y_1, y_2, \dots, y_N$ gemessen als Funktion der Unabhängigen Variable $x_1, x_2, \dots x_N$ zu fitten haben wir die Summe der Abstandsquadrate (Residuen) minimiert:

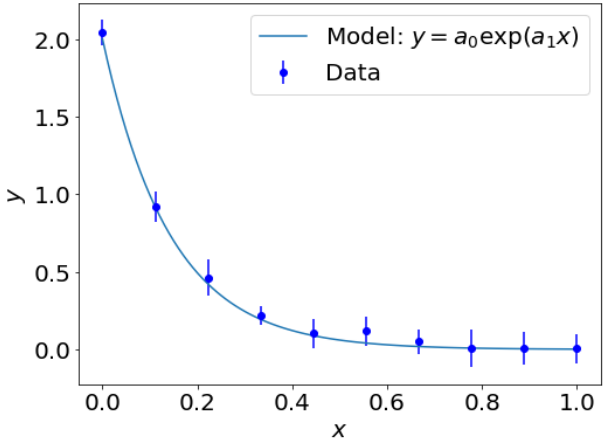
$$S = \sum_{n=1}^{N} w_n (y_n - a_0 - a_1 x_n)^2$$

Im Allgemeinen heisst das für eine Funktion $f(x, a_0, a_1, \dots, a_M) = f(x, \boldsymbol{a})$ also wir müssen

$$S = \sum_{n=1}^{N} w_n (y_n - f(x, \boldsymbol{a}))^2$$

minimieren.

Beispiel:



(Model Parameter: $a_0 = 1,\ a_1 = 5$)

Unsere Modellfunktion ist:

$y = a_0 \exp(a_1 x) = f(x, a_0, a_1)$

Nichlineare schätzung der kleinsten Quadrate

Erinnerung:

Um die lineare Funktion $a_0 - a_1 x_n$ an einen gemessenen Datensatz $y_1, y_2, \dots, y_N$ gemessen als Funktion der Unabhängigen Variable $x_1, x_2, \dots x_N$ zu fitten haben wir die Summe der Abstandsquadrate (Residuen) minimiert:

$$S = \sum_{n=1}^{N} w_n (y_n - a_0 - a_1 x_n)^2$$

Im Allgemeinen heisst das für eine Funktion $f(x, a_0, a_1, \dots, a_M) = f(x, \boldsymbol{a})$ also wir müssen

$$S = \sum_{n=1}^{N} w_n (y_n - f(x, \boldsymbol{a}))^2$$

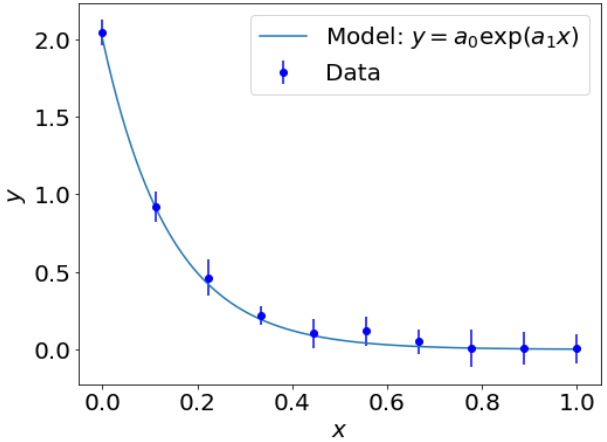
minimieren.

Wir können nun S als Funktion von „allen“ $\boldsymbol{a}$ berechnen.

Ziel: Finde das Minimum von S

Problem: Im Allgemeinen keine geschlossene Lösung – Iterative Ansatz notwendig

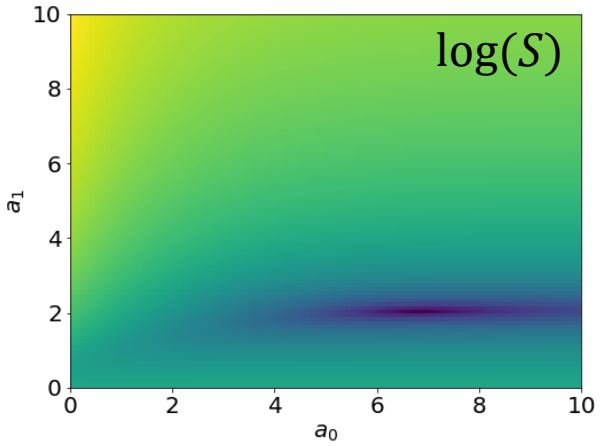
Beispiel:



(Model Parameter: $a_0 = 1,\ a_1 = 5$)

Unsere Modellfunktion ist:

$y = a_0 \exp(a_1 x) = f(x, a_0, a_1)$



1. Gradientenverfahren

Wir starten mit einer initialen Schätzung der Werte von $\boldsymbol{a}$: $\boldsymbol{a}_\mathrm{g}$ und bewegen uns auf der Fläche S entlang des steilsten Gradienten den wir aus der Linearisierung (Taylor Entwicklung erster Ordnung) erhalten. Dieser Methode folgen wir iterativ zum Minimum von S:

$$S(\boldsymbol{a}) \approx S(\boldsymbol{a}_\mathrm{g}) + \sum_{m=0}^{M} \left.\frac{\partial S}{\partial a_m}\right|_{\boldsymbol{a}_\mathrm{g}} \Delta a_m$$

Mit $\Delta a_m = a_m - a_{\mathrm{g}m}$

In Vektorschreibweise ist das:

$$S \approx S(\boldsymbol{a}_\mathrm{g}) + \Delta \boldsymbol{a}^\mathrm{T} \nabla S$$

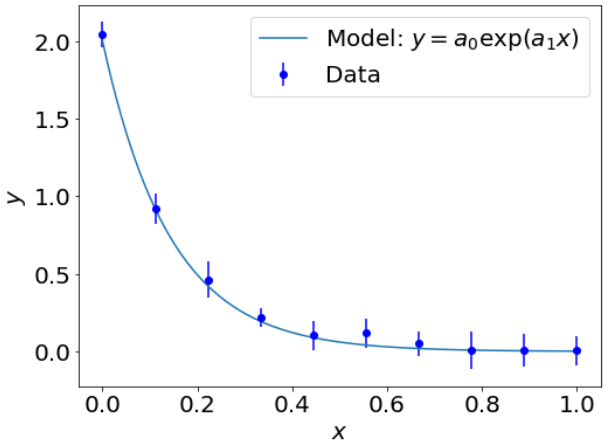
Hier ist $\nabla S = \begin{pmatrix} \frac{\partial S}{\partial a_0} \\ \vdots \\ \frac{\partial S}{\partial a_M} \end{pmatrix}$ der Gradient von S.

Damit ist die Korrektur von $\boldsymbol{a}_\mathrm{g}$ Richtung Minimum gegeben durch:

$$\Delta \hat{a}_m = -\alpha \left.\frac{\partial S}{\partial a_m}\right|_{\boldsymbol{a}_\mathrm{g}}$$

Wobei der Parameter α die „Grösse“ der Korrektur skaliert.

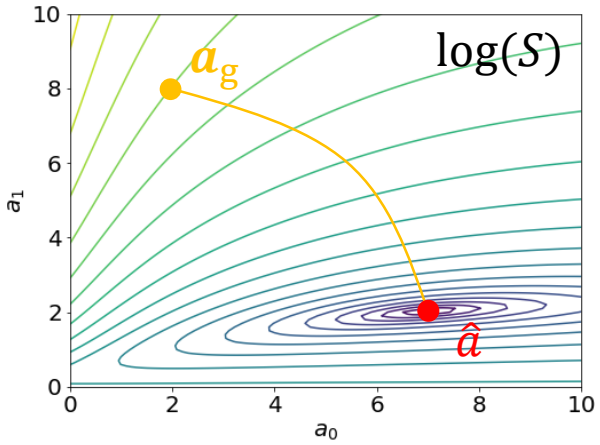
Beispiel:



(Model Parameter: $a_0 = 1,\ a_1 = 5$)

Unsere Modellfunktion ist:

$$y = a_0 \exp(a_1 x) = f(x, a_0, a_1)$$



1. Gradientenverfahren

Die Korrektur von $\boldsymbol{a}_{\mathrm{g}}$ Richtung Minimum ist gegeben durch:

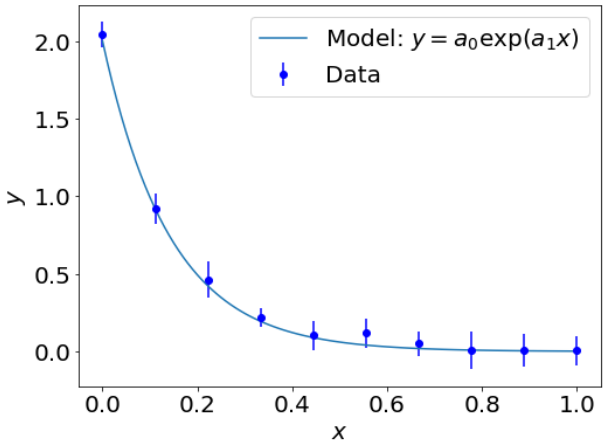
$$\Delta\hat{a}_m = -\alpha \left.\frac{\partial S}{\partial a_m}\right|_{\boldsymbol{a}_{\mathrm{g}}}$$

Damit sind die neuen Werte für die Fitparameter:

$$a_m = a_{\mathrm{g}m} + \Delta\hat{a}_m$$

Diese Prozedur wird wiederholt bis die Fitfunktion, die Daten hinreichend gut beschreibt.
Frage: Wie können wir das entscheiden?

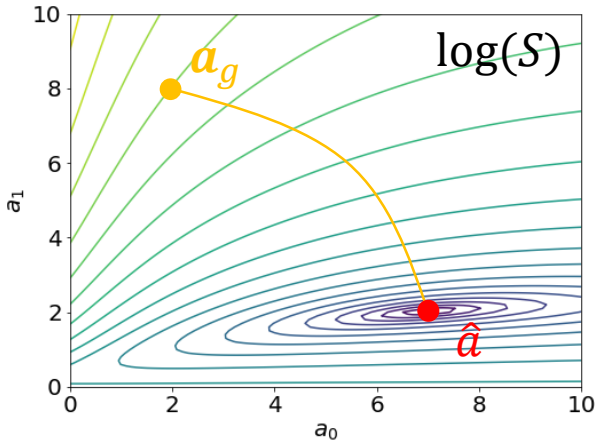
Beispiel:



(Model Parameter: $a_0 = 1,\ a_1 = 5$)

Unsere Modellfunktion ist:

$$y = a_0\exp(a_1 x) = f(x, a_0, a_1)$$



1. Gradientenverfahren

Die Korrektur von $\boldsymbol{a}_\mathrm{g}$ Richtung Minimum ist gegeben durch:

$$\Delta\hat{a}_m = -\alpha \left.\frac{\partial S}{\partial a_m}\right|_{\boldsymbol{a}_\mathrm{g}}$$

Damit sind die neuen Werte für die Fitparameter:

$$a_m = a_{\mathrm{g}m} + \Delta\hat{a}_m$$

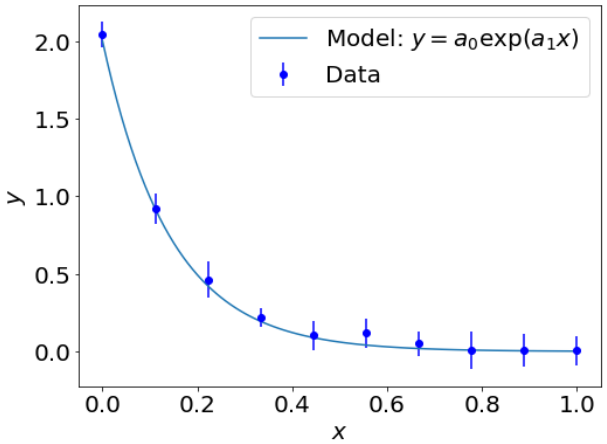
Diese Prozedur wird wiederholt bis die Fitfunktion, die Daten hinreichend gut beschreibt.
Frage: Wie können wir das entscheiden?

Wir können z.B. abbrechen wenn der Gradient sehr klein wird $\nabla S < \epsilon$. Das tatsächliche Minimum ist erreicht wenn $\nabla S = 0$

Die initialen Schätzwerte $\boldsymbol{a}_\mathrm{g}$ sowie der Parameter α müssen möglicherweise angepasst werden um die Konvergenz des Algorithmus zu erreichen.

Siehe Beispiel Python Notebook:
„ Non-Linear Fitting - Gradient Decent - single Variable“

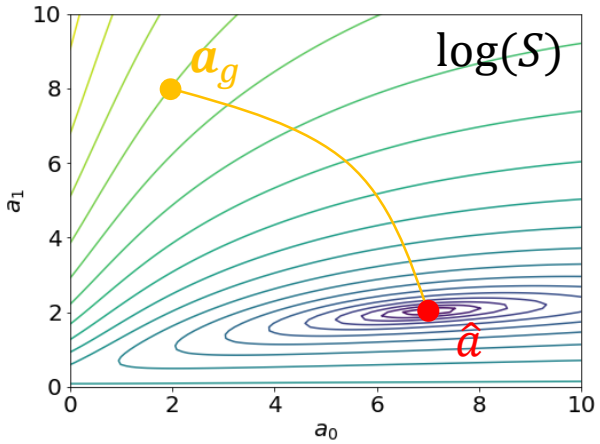
Beispiel:



(Model Parameter: $a_0 = 1,\ a_1 = 5$)

Unsere Modellfunktion ist:

$$y = a_0\exp(a_1 x) = f(x, a_0, a_1)$$



1. Gradientenverfahren

Die Korrektur von $\boldsymbol{a}_\mathrm{g}$ Richtung Minimum ist gegeben durch:

$$\Delta\hat{a}_m = -\alpha \left.\frac{\partial S}{\partial a_m}\right|_{\boldsymbol{a}_\mathrm{g}}$$

Damit sind die neuen Werte für die Fitparameter:

$$a_m = a_{\mathrm{g}m} + \Delta\hat{a}_m$$

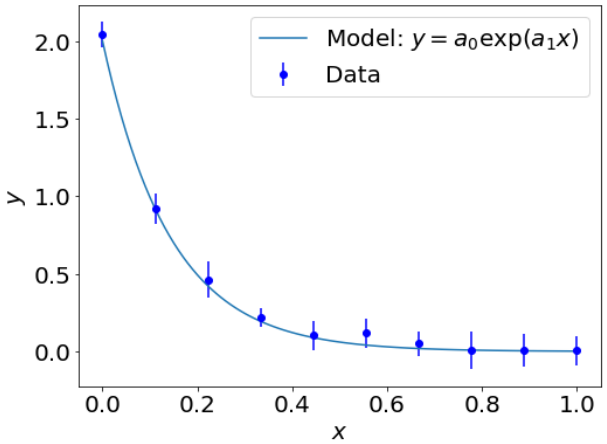
Diese Prozedur wird wiederholt bis die Fitfunktion, die Daten hinreichend gut beschreibt.
Frage: Wie können wir das entscheiden?

Wir können z.B. abbrechen wenn der Gradient sehr klein wird $\nabla S < \epsilon$. Das tatsächliche Minimum ist erreicht wenn $\nabla S = 0$

Die initialen Schätzwerte $\boldsymbol{a}_\mathrm{g}$ sowie der Parameter α müssen möglicherweise angepasst werden um die Konvergenz des Algorithmus zu erreichen.

Je näher wir dem Optimum kommen, desto kleiner werden die Schritte.
Lösung:
Anpassen von α bei jedem Schritt.

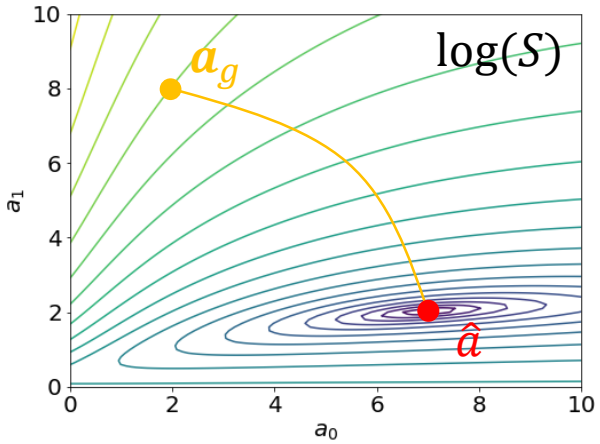
Beispiel:



(Model Parameter: $a_0 = 1,\ a_1 = 5$)

Unsere Modellfunktion ist:

$$y = a_0\exp(a_1 x) = f(x, a_0, a_1)$$



2. Newtonverfahren

Anstelle der einfachen Linearisierung von S um $\boldsymbol{a}_\mathrm{g}$ (Taylor Entwicklung erste Ordnung), können wir auch den zweiten Term der Taylor Entwicklung berücksichtigen:

$$S \approx S(\boldsymbol{a}_\mathrm{g}) + \sum_{m=0}^{M} \left.\frac{\partial S}{\partial a_m}\right|_{\boldsymbol{a}_\mathrm{g}} \Delta a_m + \frac{1}{2}\sum_{m=0}^{M}\sum_{k=0}^{M} \left.\frac{\partial^2 S}{\partial a_m \partial a_k}\right|_{\boldsymbol{a}_\mathrm{g}} \Delta a_m \Delta a_k = S'(\boldsymbol{a})$$

Wir approximieren S also durch einen Paraboloid. Der nächste Iterationsschritt ist gegeben durch das Minimum von S'.

Das bedeutet das für alle Δa_l: $\frac{\partial S'}{\partial \Delta a_l} = 0$:

$$\left.\frac{\partial S}{\partial a_l}\right|_{\boldsymbol{a}_\mathrm{g}} + \sum_{m=0}^{M} \left.\frac{\partial^2 S}{\partial a_l \partial a_m}\right|_{\boldsymbol{a}_\mathrm{g}} \Delta \hat{a}_m = 0$$

Hier ist $\hat{\boldsymbol{a}}$ der Vektor mit den Korrekturen für die initialen Parameter.

Dies können wir in Matrixschreibweise schreiben:

$$\begin{pmatrix} \frac{\partial^2 S}{\partial a_0^2} & \cdots & \frac{\partial^2 S}{\partial a_0 \partial a_M} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 S}{\partial a_M \partial a_0} & \cdots & \frac{\partial^2 S}{\partial a_M^2} \end{pmatrix} \begin{pmatrix} \Delta \hat{a}_0 \\ \vdots \\ \Delta \hat{a}_M \end{pmatrix} = - \begin{pmatrix} \frac{\partial S}{\partial a_0} \\ \vdots \\ \frac{\partial S}{\partial a_M} \end{pmatrix}$$

2. Newtonverfahren

Anstelle der einfachen Linearisierung von S um $\boldsymbol{a}_\mathrm{g}$ (Taylor Entwicklung erste Ordnung), können wir auch den zweiten Term der Taylor Entwicklung berücksichtigen:

$$S \approx S(\boldsymbol{a}_\mathrm{g}) + \sum_{m=0}^{M} \left.\frac{\partial S}{\partial a_m}\right|_{\boldsymbol{a}_\mathrm{g}} \Delta a_m + \frac{1}{2}\sum_{m=0}^{M}\sum_{k=0}^{M} \left.\frac{\partial^2 S}{\partial a_m \partial a_k}\right|_{\boldsymbol{a}_\mathrm{g}} \Delta a_m \Delta a_k = S'(\boldsymbol{a})$$

Dies können wir in Matrixschreibweise schreiben:

$$\begin{pmatrix} \frac{\partial^2 S}{\partial a_0^2} & \cdots & \frac{\partial^2 S}{\partial a_0 \partial a_M} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 S}{\partial a_M \partial a_0} & \cdots & \frac{\partial^2 S}{\partial a_M^2} \end{pmatrix} \begin{pmatrix} \Delta \hat{a}_0 \\ \vdots \\ \Delta \hat{a}_M \end{pmatrix} = - \begin{pmatrix} \frac{\partial S}{\partial a_0} \\ \vdots \\ \frac{\partial S}{\partial a_M} \end{pmatrix}$$

Hesse Matrix $\boldsymbol{H}$ Gradient $\boldsymbol{\nabla} S$

Und auch der Ausdruck für S':

$$S' = S(\boldsymbol{a}_\mathrm{g}) + \Delta \boldsymbol{a}^\mathrm{T} \boldsymbol{\nabla} S + \frac{1}{2} \Delta \boldsymbol{a}^\mathrm{T} \boldsymbol{H} \Delta \boldsymbol{a}$$

2. Newtonverfahren

Das Gleichungssystem für die Nullstellenfindung von S in Matrixschreibweise ist:
(Wobei alle Ableitungen bei $\boldsymbol{a}_{\mathrm{g}}$ ausgewertet werden)

$$\begin{pmatrix} \frac{\partial^2 S}{\partial a_0^2} & \cdots & \frac{\partial^2 S}{\partial a_0 \partial a_M} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 S}{\partial a_M \partial a_0} & \cdots & \frac{\partial^2 S}{\partial a_M^2} \end{pmatrix} \begin{pmatrix} \Delta \hat{a}_0 \\ \vdots \\ \Delta \hat{a}_M \end{pmatrix} = - \begin{pmatrix} \frac{\partial S}{\partial a_0} \\ \vdots \\ \frac{\partial S}{\partial a_M} \end{pmatrix}$$

Hesse Matrix $\boldsymbol{H}$ Gradient $\boldsymbol{\nabla} S$

Also:

$$\boldsymbol{H}\, \Delta \hat{\boldsymbol{a}} = -\boldsymbol{\nabla} S$$

Und damit erhalten wir für die Korrekturen zu den initialen Parametern:

$$\Delta \hat{\boldsymbol{a}} = -\boldsymbol{H}^{-1} \boldsymbol{\nabla} S$$

Die neuen Parameter sind dann: $\hat{\boldsymbol{a}} = \boldsymbol{a}_{\mathrm{g}} + \Delta \hat{\boldsymbol{a}}$

Auch hier fügen wir für bessere Konvergenz einen Skalierungsfaktor ($\alpha < 1$) ein:

$$\hat{\boldsymbol{a}} = \boldsymbol{a}_{\mathrm{g}} + \alpha \Delta \hat{\boldsymbol{a}}$$

Siehe Beispiel Python Notebook:
„Non-Linear Fitting - Gradient Decent - single Variable“

Vergleich der beiden Methoden

Wir berechnen die Korrektur als:

- Gradientenverfahren:

$$\Delta\hat{\boldsymbol{a}} = -\alpha\boldsymbol{\nabla}S$$

- Newtonverfahren:

$$\Delta\hat{\boldsymbol{a}} = -\alpha\boldsymbol{H}^{-1}\boldsymbol{\nabla}S$$

Eine intuitive und einfache Verbesserung der Methode des Gradientenverfahren ist also α dynamisch anzupassen, in dem wir es mit der inversen Krümmung von S skalieren:

$$\Delta\hat{a}_m = -\alpha\left(\frac{\partial^2 S}{\partial a_m^2}\right)^{-1}\frac{\partial S}{\partial a_m}$$

Grundsätzlich ist es wünschenswert die Vorteile von beiden Methoden zu kombinieren und ihre Nachteile zu kompensieren.

3. Marquardts Methode

Wir wollen die Verfahren 1 und 2 kombinieren um schnelle Konvergenz in allen Regimen zu erreichen.

- Wenn wir weit weg sind vom Optimum (wenn die 2te Ableitung von S gross ist) wollen wir ein konstantes α benutzen.
- Wenn wir dem Optimum näher kommen (wenn die 2te Ableitung von S klein wird) möchten wir die Korrektur mit der inversen Hesse Matrix skalieren.

Eine Methode das zu erreichen wurde von D. Marquardt vorgeschlagen, in dem die Matrix $\boldsymbol{H}$ ersetzt wird durch:

$$\boldsymbol{H}'_{i,j} = \begin{cases} \boldsymbol{H}_{ij}(1+\alpha) & i = j \\ \boldsymbol{H}_{ij} & i \neq j \end{cases}$$

Wir benutzen $\boldsymbol{H}'$ um $\hat{\boldsymbol{a}}$ zu berechnen:

$$\Delta\hat{\boldsymbol{a}} = -\left(\boldsymbol{H}'\right)^{-1}\boldsymbol{\nabla}S$$

Falls α sehr gross wird vereinfacht sich dies zu:

$$\Delta\hat{a}_m = \frac{1}{(1+\alpha)}\left(\frac{\partial^2 S}{\partial a_m^2}\right)^{-1}\frac{\partial S}{\partial a_m}$$

Das ist sehr ähnlich wie unsere einfache Verbesserung des Gradientenverfahrens.

Siehe Beispiel Python Notebook:
„Non-Linear Fitting - Gradient Decent - final - Vorlesung - Teil 2“

Donald W. Marquardt, *An Algorithm for Least-Squares Estimation of Nonlinear Parameters*
Journal of the Society for Industrial and Applied Mathematics **11**, 431 (1963)

Unsicherheit der Fitparameter

Mit diesen Methoden erhalten wir nicht automatisch die Kovarianzmatrix aus der Normalform, wie bei der linearen Minimierung der Summer der Abstandsquadrate.

Aber wir können die Kovarianzmatrix aus der Hesse Matrix berechnen:

Hierzu nehmen wir an, dass wir die Werte $\hat{\boldsymbol{a}}$, die S minimieren, mit einer der vorher beschriebenen Methoden bestimmt haben:

$$S_{\min} = \sum_{n=0}^{N} w_n \left(y_n - f(x_n, \hat{\boldsymbol{a}})\right)^2$$

Wir linearisieren f um $\hat{\boldsymbol{a}}$ mittels einer Taylor Entwicklung erster Ordnung:

$$f(x, \boldsymbol{a}) \approx f(x, \hat{\boldsymbol{a}}) + \sum_{k=0}^{M} \left.\frac{\partial f(x, \boldsymbol{a})}{\partial a_k}\right|_{\hat{\boldsymbol{a}}} (a_k - \hat{a}_k) = f(x, \hat{\boldsymbol{a}}) + \sum_{k=0}^{M} \frac{\partial f(x, \boldsymbol{a})}{\partial a_k} \Delta a_k$$

Mit $\Delta a_m = a_m - \hat{a}_m$ und von nun an werden alle Ableitungen an der Position $\hat{\boldsymbol{a}}$ ausgewertet.

Hiermit können wir S abschätzen als:

$$S \approx \sum_{n=0}^{N} w_n \left(y_n - f(x_n, \hat{\boldsymbol{a}}\,) - \sum_{m=0}^{M} \frac{\partial f(x_n, \boldsymbol{a})}{\partial a_m} \Delta a_m \right)^2$$

Unsicherheit der Fitparameter

Durch Linearisierung von f um $\hat{\boldsymbol{a}}$ können wir S abschätzen als:

$$S \approx \sum_{n=0}^{N} w_n \left(y_n - f(x_n, \hat{\boldsymbol{a}}) - \sum_{m=0}^{M} \frac{\partial f(x_n, \boldsymbol{a})}{\partial a_m} \Delta a_m \right)^2 =$$

$$= \underbrace{\sum_{n=0}^{N} w_n \left(y_n - f(x_n, \hat{\boldsymbol{a}})\right)^2}_{= S_{\min}} - 2 \underbrace{\sum_{n=0}^{N} \left(w_n (y_n - f(x_n, \hat{\boldsymbol{a}})) \sum_{m=0}^{M} \frac{\partial f(x_n, \boldsymbol{a})}{\partial a_m} \Delta a_m \right)}_{= \left. \frac{\partial S}{\partial \Delta a_m} \right|_{\Delta a_m = 0} = 0} + \underbrace{\sum_{n=0}^{N} w_n \left(\sum_{m=0}^{M} \frac{\partial f(x_n, \boldsymbol{a})}{\partial a_m} \Delta a_m \right)^2}_{= \sum_{m=0}^{M} \frac{\partial f(x_n, \boldsymbol{a})}{\partial a_m} \Delta a_j \sum_{k=0}^{M} \frac{\partial f(x_n, \boldsymbol{a})}{\partial a_k} \Delta a_k} =$$

$$= S_{\min} + \sum_{m=0}^{M} \sum_{k=0}^{M} \Delta a_m \Delta a_k \sum_{n=0}^{N} w_n \frac{\partial f(x_n, \boldsymbol{a})}{\partial a_m} \frac{\partial f(x_n, \boldsymbol{a})}{\partial a_k} =$$

$$= S_{\min} + \Delta \boldsymbol{a}^{\mathrm{T}} N \Delta \boldsymbol{a}$$

Unsicherheit der Fitparameter

Durch Linearisierung von f um $\widehat{\boldsymbol{a}}$ können wir S abschätzen als:

$$S = S_{\min} + \Delta \boldsymbol{a}^{\mathrm{T}} \boldsymbol{N} \Delta \boldsymbol{a}$$

So haben wir das nichtlineare Problem auf ein lineares reduziert und können die Methoden anwenden, die wir für den linearen Fall etabliert haben. Somit können wir im Prinzip die Kovarianzmatrix $\boldsymbol{C} = \boldsymbol{N}^{-1}$ berechnen.

Um die explizite Berechnung von $\boldsymbol{N}$ zu vermeiden können wir uns erinnern, dass:

$$S = S(\boldsymbol{a}_{\mathrm{g}}) + \Delta \boldsymbol{a} \boldsymbol{\nabla} S + \frac{1}{2} \Delta \boldsymbol{a}^{\mathrm{T}} \boldsymbol{H} \Delta \boldsymbol{a}$$

Am Minimum von S, wo $\boldsymbol{a}_{\mathrm{g}} = \widehat{\boldsymbol{a}}$ und $\boldsymbol{\nabla} S = 0$ wird das zu:

$$S = S_{\min} + \frac{1}{2} \Delta \boldsymbol{a}^{\mathrm{T}} \boldsymbol{H} \Delta \boldsymbol{a}$$

Und somit $\frac{1}{2} \Delta \boldsymbol{a}^{\mathrm{T}} \boldsymbol{H} \Delta \boldsymbol{a} = \Delta \boldsymbol{a}^{\mathrm{T}} \boldsymbol{N} \Delta \boldsymbol{a}$, was bedeutet, dass $\boldsymbol{N} = \frac{1}{2} \boldsymbol{H}$

Wir können also die Kovarianzmatrix aus der Hesse Matrix berechnen:

$$\boldsymbol{C} = 2 \boldsymbol{H}^{-1}$$

Siehe Beispiel Python Notebook:
„Non-Linear Fitting - Gradient Decent - final – Vorlesung – Teil 2“

Anwendung der Python Methode SciPy Curve Fit

In Python wie auch in den meisten anderen Datenanalysewerkzeugen gibt es effiziente und zuverlässige Methoden zur nichtlinearen Kurvenanpassung.
In Python z.B. in der Bibliothek SciPy:

Importieren der Funktion:

```
from scipy.optimize import curve_fit
```

Definiere ein Modell das gefittet werden soll:

```
def expmodel(x, amplitude, gamma):
    return amplitude*np.exp(-gamma*x)

popt, pcov = curve_fit(expmodel, x, y, sigma = np.sqrt(1/w), method='lm', absolute_sigma=True)
```

Ergebnisse:
`popt` – Optimale Fitparameter.
`pcov` – Covarianzmatrix. Achtung, wenn nicht explizit `absolute_sigma=True` angegeben wird, wird die Kovarianzmatrix folgendermassen berechnet:

$$\boldsymbol{C} = \frac{\hat{S}_{\min}}{N - M - 1} \boldsymbol{N}^{-1}$$

Zusammenfassung

- Wir können für jede Funktion die Kostenfunktion S (die Summe der Abstandsquadrate) definieren.
- Minimieren von S erlaubt uns die Funktionsparameter zu finden für die die Funktion die Daten am besten beschreibt.
- Dazu folgen wir iterativ dem Gradienten von S zum Minimum.
- Die Unsicherheit der Fitparameter können wir mit Hilfe der Hesse-Matrix berechnen.
- Die meisten Datenanalyse Werkzeuge stellen effiziente und zuverlässige Implementation dieser Methoden zu Verfügung. In Python z.B. lmfit