Numerische Methoden mit SciPy
Ableitungen, Extremwerte, Curve Fitting & Fehleranalyse

Milan Ončák, Gabriel Schöpfer, Michael Gatt, Michael Hütter

30/09/2025

Agenda

1. SciPy Überblick
2. Basics: Ableitungen
3. Basics: Extremwerte
4. Basics: Curve Fitting
5. Fehleranalyse & Unsicherheitsquantifizierung
6. Praxisleitlinien
7. Übungen

SciPy Überblick

SciPy in der Datenverarbeitung

- **NumPy**: Arrays, Broadcasting, lineare Algebra
- **SciPy**: High-Level-Algorithmen auf NumPy: *`optimize`*, *`interpolate`*, *`signal`*, *`stats`*
- **Matplotlib**: Visualisierung
- **Pandas**: Tabellen/IO

Philosophie: „Don't reinvent the wheel - just import what you need."

Typischer Workflow

1. Daten laden/erzeugen (`numpy.loadtxt`, `pandas.read_csv`)
2. Vorverarbeitung (Filter, Normierung, usw.)
3. Modell formulieren (Physik!), Kostenfunktion definieren
4. Optimierung/Fit (`optimize.minimize`, `optimize.curve_fit`)
5. Gütebewertung: Residuen, χ^2, Kovarianzmatrix

Basics: Ableitungen

Ableitungen

$$d_x f(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

- Symbolisch (`SymPy`)
- Numerisch (`NumPy/SciPy`)
- AutoDiff (`jax`)

Numerische Ableitungen: Idee

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}, \quad h \equiv \text{kleine Zahl}$$

Fehlerquellen: Diskretisierung ($\propto h$), Rundungsfehler ($\propto 1/h$). Trade-off wählen!

- Gradient: Finite Differenzen komponentenweise

SciPy-Tools für Ableitungen

- NumPy/SciPy = Finite Differenzen
- SciPy: scipy.optimize.approx_fprime()

```
import numpy as np
from scipy.optimize import approx_fprime

f = lambda x: np.sin(x[0]) + x[1]**2
x0 = np.array([1.0, 2.0])

grad = approx_fprime(x0, f)
```

SciPy-Tools für Ableitungen

- Ableitung von Datenpunkten?
- z.B. Sensordaten
- NumPy: numpy.gradient() (CD)
- Einfache Datensätze können auch direkt mittels np.loadtxt() in einen np.array eingelesen werden

```
barometer = np.loadtxt("data_new/Barometer.csv", delimiter=",",
    skiprows=1)
t = barometer[:, 0]
h = barometer[:, 1]
v_vertical = np.gradient(h,t)
```

Basics: Extremwerte

SciPy-Tools für Extremwerte

- Nichtlineare Optimierung! Im allgemeinen ein sehr komplexes Thema
- Globales vs. lokales Minimum $\Rightarrow$ Startwerte sind wichtig!
- Konvexität macht das Leben einfach (z.B. Quadratische Fehlerfunktionen)
- Nebenbedingugen können automatisch berücksichtigt werden
- SciPy: optimize.minimize
- optimize.minimize_scalar

```
f = lambda x: (x[0] - 2)**2 + 1
x0 = np.array([0.0])
res = optimize.minimize(f, x0)
```

Basics: Curve Fitting

Least Squares: Prinzip

Gegeben Daten (x_i, y_i, σ_i) und Modell $y = F(x; \theta)$.

$$\chi^2(\theta) = \sum_i \left(\frac{y_i - F(x_i; \theta)}{\sigma_i} \right)^2 \quad \Rightarrow \quad \hat{\theta} = \arg\min \chi^2$$

```
def F(x, a, b):
    return a * x + b
popt, pcov = curve_fit(F, x, y, [1, 1])
a, b = popt
```

x_i unabhängige Variable (z. B. Zeit, Temperatur, …)
y_i gemessene Zielgröße zu x_i
σ_i Unsicherheiten (Standardabweichung) zu y_i

Least Squares: Gängige Probleme

- Modell nicht linear in Parametern $\Rightarrow \chi^2$ eine komplizierte Funktion
- Nicht-konvex $\Rightarrow$ Globales Minimum nicht garantiert
- Gute Anfangswerte der Parameter wichtig!!!

Least Squares: Gängige Probleme

- Annahme: unabhängige, normalverteilte Residuen $r_i = y_i - F(x_i, \hat{\theta})$
- Unabhängig: Keine systematischen Korrelationen
- Normalverteilung: $r_i \sim \mathcal{N}(0, \sigma_i^2)$ (Maximum-Likelihood-Problem äquivalent zum Minimieren von χ^2)
- Homoskedastizität: Alle Residuen haben die selbe Varianz σ^2

Dies ist wichtig für die Fehleranalyse, denn nur unter diesen Annahmen sind Standardabweichungen der Fit-Parameter aus der Kovarianzmatrix gültig.

$$\sigma^2(X) = \mathbb{E}[(X - \mu)^2]$$

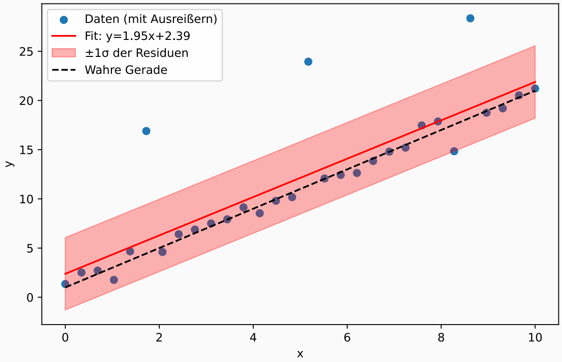
$$\mathrm{Cov}(X, X) = \mathbb{E}[(X - \mu)(X - \mu)^T], \quad \mu = \mathbb{E}(X), \;\; X = (X_1, X_2, ..., X_n)^T$$

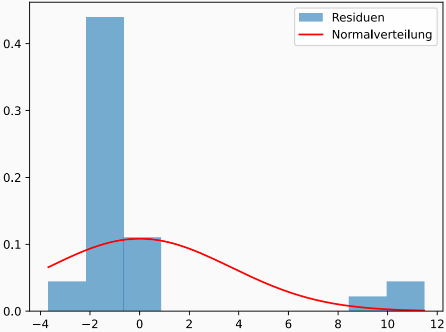
Beispiel: Korrelierte Residuen

- Oftmals in Zeitreihenexperimenten (z.B. Sensordrift)

Beispiel: Verletzung der Normalverteilung

- Einzelne Messungen stark fehlerhaft





Fehleranalyse & Unsicherheitsquantifizierung

Messfehler

- Statistische Streuung der Daten
- Rauschen im Messgerät, zufällige Schwankungen der Umwelt, ...

$$\sigma_{\text{resid}} = \sqrt{\frac{1}{N-p}\sum_i r_i^2}, \quad p = \text{Anzahl der geschätzten Parameter}$$

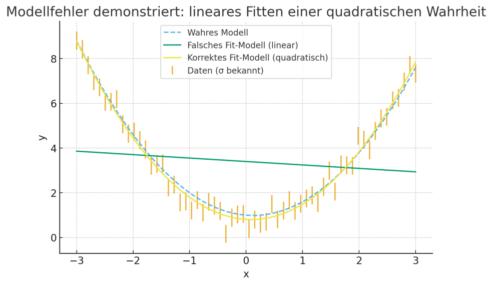
- Systematische Fehler könen nur durch Vergleich mit Referenzexperimenten oder Kalibrierungskurven berücksichtigt werden!

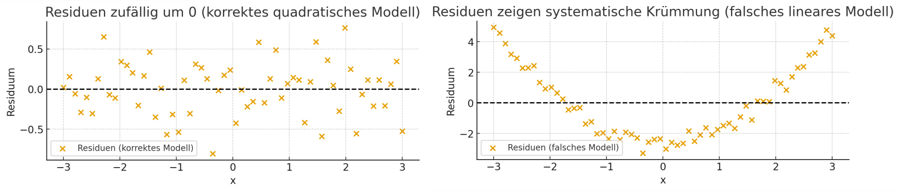
R^2

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2} \tag{1}$$

- Wie gut erklärt ein Modell die Streuung der beobachteten Daten
- Im Normalfall zwischen 0 und 1
- Mehr Prädikatoren erhöhen R^2 fast immer (ohne echten Nutzen)
- Misst Güte der Anpassung, nicht Vorhersageleistung

Modellfehler (falsches Fit-Modell)





$$\chi^2_{\nu;\mathrm{quad}} \approx 0.717 \qquad \chi^2_{\nu;\mathrm{lin}} \approx 35.28$$

Schätzungsfehler der Parameter

- Endliche Anzahl von Datenpunkten $\Rightarrow$ der Fit ist eine Schätzung
- Kann mittels Kovarianzmatrix aus Least Squares (LS) quantifiziert werden

$$\mathrm{cov}(\hat{a}, \hat{b}) = \sigma^2_{\mathrm{resid}}(X^T X)^{-1}$$

$$\Delta a = \sqrt{\mathrm{cov}(a, a)}$$

- Achtung: Nur korrekt, wenn Residuen unabhängig & normalverteilt sind!
- SciPy: ***pcov*** = Kovarianzmatrix der Parameter
- Diagonalelemente = Varianz der Schätzung
- Wurzel daraus = Standardfehler = 1σ-Unsicherheiten

Schätzungsfehler der Parameter

```python
import numpy as np
from scipy.optimize import curve_fit

def lin(x, a, b):
    return a*x + b

x = ...
y = ...

popt, pcov = curve_fit(lin, x, y)
a, b = popt

# Standardfehler = Wurzel der Diagonale der Kovarianzmatrix
perr = np.sqrt(np.diag(pcov))
da, db = perr
```

Numerische Fehler

- Endliche Maschinenpräzision, Rundungsfehler, schlecht konditionierte Designmatrix
- Achtung: Besonders kritisch bei stark korrelierten x-Werten, schlechte Skalierung der Daten, hohe Polynomgrade

Praxisleitlinien

Best Practices

- Unsicherheiten **immer** mitführen (Gewichtung, Fehlerbalken)
- Startwerte bei nichtlinearen Fits sorgfältig wählen
- Skaliere Parameter/Variablen (Kondition)
- Dokumentiere Annahmen, Einheiten, Vorverarbeitung

Typische Fallstricke

- Verwechslung von *Standardabweichung* und *Standardfehler*
- Unterschätzung von Unsicherheiten bei korreliertem Rauschen
- Overfitting durch zu flexible Modelle
- Ignorieren von Ausreißern

Fragen?

Übungen

Übung: Ableitung 1

Gegeben: Die Funktion

$$f(x,y) = \sin(x) \cdot \cos(y).$$

Aufgaben:

1. Verwende *`scipy.optimize.approx_fprime()`*, um den Gradienten $\nabla f(x,y)$ an der Stelle $(x,y) = (\frac{\pi}{4}, \frac{\pi}{3})$ numerisch zu berechnen.
2. Vergleiche das Ergebnis mit dem exakten Gradienten

$$\nabla f(x,y) = \big(\cos(x)\cos(y),\ -\sin(x)\sin(y)\big).$$

Die Datei *Barometer.csv* enthält Messdaten eines Paragleitfluges.

- *seconds_elapsed* – Zeit in Sekunden
- *relativeAltitude* – relative Höhe in Metern

Aufgaben:

1. Lade die Daten ein und stelle den Höhenverlauf grafisch dar.
2. Berechne die Vertikalgeschwindigkeit mit Hilfe von *np.gradient()*.
3. Bestimme die maximale Steig- und Sinkgeschwindigkeit mit *np.max()*, *np.min()*.
4. Stelle die Vertikalgeschwindigkeit als Funktion der Zeit dar.

Diskussionsfragen:

- Warum verstärkt eine Ableitung vorhandenes Rauschen? Was könnte man dagegen tun?

- Implementiere eine zentrale Differenzenquotienten-Formel zur Approximation der Ableitung:

$$d_x f(x) \approx \frac{f(x+h) - f(x-h)}{2h}.$$

- Untersuche die Genauigkeit für $f(x) = \sin(x)$ im Intervall $[0, 2\pi]$, indem du den Gesamtfehler in Abhängigkeit von der Schrittweite h darstellst.
- Sind die voreingestellten Parameter von *`scipy.optimize.approx_fprime()`* geeignet?

Übung: Extremwerte

Gegeben: Ein Körper wird mit $v_0 = 40\,\mathrm{m/s}$ unter einem Winkel θ abgeworfen. Ohne Luftwiderstand gilt für die Reichweite:

$$R(\theta) = \frac{v_0^2}{g}\sin(2\theta).$$

Aufgaben:

1. Implementiere $R(\theta)$ in Python.
2. Verwende *`scipy.optimize.minimize_scalar`*, um den Winkel $\theta^\star$ zu bestimmen, der die Reichweite maximiert.
   - Tipp: Maximiere $R(\theta)$, indem du $-R(\theta)$ minimierst.
3. Vergleiche den gefundenen Wert mit der theoretischen Vorhersage $\theta^\star = 45°$.

Übung: Exponentieller Zerfall & Fehleranalyse

Eine Person behauptet, die Datei *`exp_decay.csv`* enthalte reale Messungen des radioaktiven Zerfalls einer Uranprobe.

- Fitte das Modell $y(t) = A\,e^{-t/\tau} + B$ mit *`scipy.optimize.curve_fit`* (**gewichtet mit** $\sigma_i \approx \sqrt{y_i}$) und bestimme Parameter, Standardfehler und die geschätzte Halbwertszeit $t_{1/2} = \tau \ln 2$.
- Erstelle Residuenplots und ein Histogramm der standardisierten Residuen $z_i = r_i/\sigma_i$.
- Quantifiziere die Streuung über das reduzierte χ^2.
- Beurteile: Kann es sich tatsächlich um eine echte Uranprobe handeln? Wenn ja, um welches Isotop handelt es sich?

Hinweis: Zeitachse in Minuten; Spalte *`counts_per_min`* enthält Zählraten pro Minute.

Bonusaufgabe: Splines

Ziel: Aus den Höhenmessungen eines Paragleitflugs eine robuste Vertikalgeschwindigkeit ableiten mittels splines.

Aufgaben:

1. Fitte einen glättenden kubischen Spline $S(t)$ auf (t, h) (`scipy.interpolate.UnivariateSpline`).
2. Bestimme die Vertikalgeschwindigkeit als $v_z^{(\mathrm{spline})}(t) = S'(t)$ und vergleiche sie mit der Roh-Ableitung $v_z^{(\mathrm{raw})}(t) = \mathrm{gradient}(h, t)$.
3. Identifiziere maximale Steig- und Sinkgeschwindigkeit für beide Methoden.
4. Variiere den Glättungsparameter s (z. B. logarithmische Skala) und diskutiere den Trade-off: Rauschen vs. Glättung (Bias–Varianz).

Anhang

SciPy-Tools für Extremwerte

- Nichtlineare Optimierung! Im allgemeinen ein sehr komplexes Thema
- Globales vs. lokales Minimum $\Rightarrow$ Startwerte sind wichtig!
- Konvexität macht das Leben einfach (z.B. Quadratische Fehlerfunktionen)
- Nebenbedingugen können automatisch berücksichtigt werden
- SciPy: optimize.minimize
  - BFGS, keine Bounds/Constraints
  - L-BFGS-B, Bounds aber keine Constraints
  - SLSQP, Constraints
- optimize.minimize_scalar

```
f = lambda x: (x[0] - 2)**2 + 1
x0 = np.array([0.0])
res = optimize.minimize(f, x0)
```

BFGS, SLSQP, ...???

- Grundidee Iterative minimierung einer quadratischen approximation in t

$$f(x_k + t) \approx f(x_k) + f'(x_k)t + \frac{1}{2}f''(x_k)t^2$$

$$0 = d_t[f(x_k) + f'(x_k)t + \frac{1}{2}f''(x_k)t^2] = f'(x_k) + f''(x_k)t$$

$$\Rightarrow t = -\frac{f'(x_k)}{f''(x_k)}$$

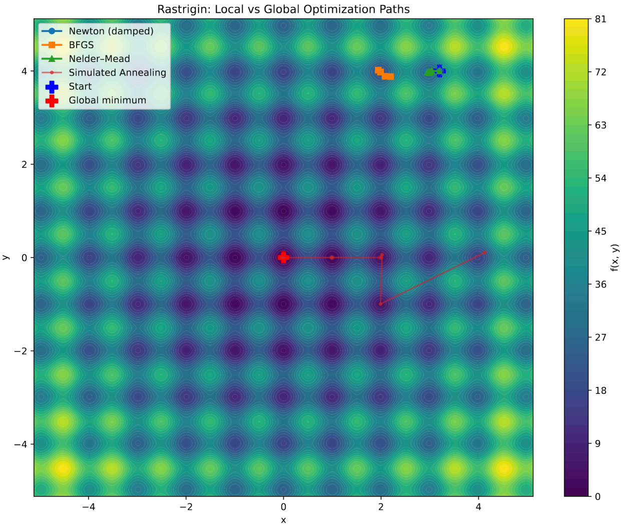
$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$$

BFGS, SLSQP, ...???

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}$$

- Newton-Methode
- Gradientenverfahren: nur erste Ableitung
- Quasi-Newton (BFGS, L-BFGS-B): approximation von H (f'')
- Nelder-Mead: keine Ableitung
- Globale Optimierung z.B. mit simulated annealing oder genetic algorithms möglich

Extremwerte: Praktisches Beispiel



$1/N$, oder doch besser $1/(N-1)$? Empirische Varianz

$$\sigma^2 = \frac{1}{N}\sum_i (y_i - \mu)^2, \qquad \mu = \text{Erwartungswert}$$

- Problem: Wir schätzen nur das empirische Stichprobenmittel $\bar{\mu}$
- Das Stichprobenmittel minimiert die Residuenquadratsumme
- Varianz wird unterschätzt

Modellfehler (falsches Fit-Modell)

- Man nimmt ein lineares Modell, obwohl die wahre Beziehung nicht linear ist.
- Quantifizierung durch Residuenplot (systematische Abweichungen kann ein Hinweis auf Modellfehler sein)
- Erhöter reduzierter χ^2

$$\chi_\nu^2 = \frac{\chi^2}{N - p} \gg 1$$

- Achtung: Ein scheinbar guter Fit (hohes R^2) kann trotzdem ein falsches Modell sein!