Numerische Methoden der Physik Serie 3

Cedric Sigrist

14. April 2025

1 Vorbereitung & Ausreisserdetektion

Die bisher implementierte Ausreisserdetektion geht davon aus, dass die Daten eindimensional und ungefähr normalverteilt sind. Dabei werden Werte, die höchstwahrscheinlich nicht zur Normalverteilung gehören als Outlier erkannt (z.B. mehr als 4σ Abweichung). Diese Annahme trifft hier allerdings nicht zu, da es sich um zweidimensionale, stark korrelierte Daten handelt.

Es gibt verschiedene Ansätze, um mit solchen Daten umzugehen. Ich habe mich entschieden, die Ausreisserdetektion separat auf die Zeit und Meeresspiegeldaten anzuwenden. Dabei wird allerdings komplett ignoriert, dass diese Korreliert sind. Entsprechend können nur Datenpunkte erkannt werden, die stark von der Gesamtheit aller Messungen in einer der beiden Dimensionen abweichen.

Eine alternative Herangehensweise wäre, ein Modell zu fitten und Datenpunkte mit grossen Residuen als Ausreisser zu betrachten. In der Aufgabenstellung wird jedoch explizit verlangt die Daten vorab der Modellierung zu bereinigen.

Mit der Gewählten Ausreisserdetektion wurden keine Ausreisser in den Daten gefunden.

2 Mittelwert (n=0)

Macht man einen Linearer Ausgleich der Daten, 0-ter Ordnung, bestimmt also den Parameter

$$\underline{y} \approx x_0 \qquad \Leftrightarrow \qquad \underline{l} \approx Ax_0$$

mit

$$A = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$$

Mit der Normalengleichung (mit Gewichtung $P = \mathbb{1}$) erhält man $x_0 = 11.4$. Mit den Residuen lässt sich nun $m_0 = 29.0$ berechnen. Vergleicht man diese Werte mit dem Mittelwert $\bar{l} = 11.4$ und der Standardabweichung $\sigma_l = 29.0$ des Meeresspiegels, findet man dass sie übereinstimmen. Die Resultate sind in Abbildung 2.1 dargestellt.

2.1 Mathematischer Zusammenhang

Der Zusammenhang zwischen den berechneten Werten wird ersichtlich wenn man deren Formeln vergleicht.

$$x_0 = (A^\mathsf{T}A)^{-1}A^\mathsf{T}\underline{l} = \frac{1}{n}\sum_{i=1}^{n} l_i = \bar{l}$$

$$m_0^2 = \frac{v^\mathsf{T}v}{n-u} = \frac{(\underline{l}-x_0)^\mathsf{T}(\underline{l}-x_0)}{n-1} = \frac{1}{n-1}\sum_{i=1}^{n}(l_i-\bar{l})^2 = \sigma_l^2$$

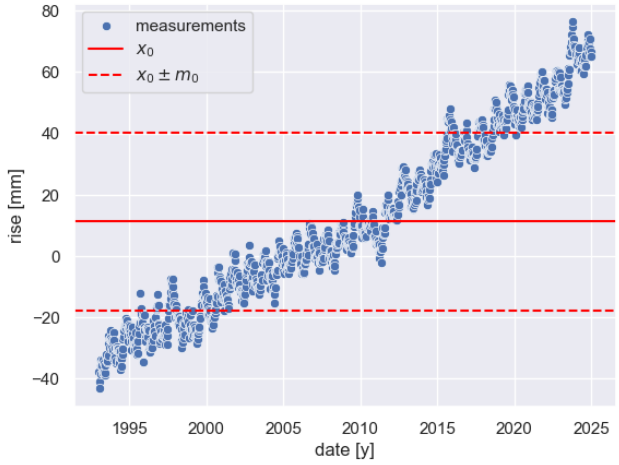


Abbildung 2.1: Linearer Ausgleich mit Polynom 0-ter Ordnung

3 Regressionsgerade (n=1)

Hier wurden die Parameter wieder mit der Normalengleichung bestummen. Man erhält

$$\begin{bmatrix} x_1 \\ x_0 \end{bmatrix} = \begin{bmatrix} 3.073\,\mathrm{mm/y} \\ -6.164 \times 10^3\,\mathrm{mm} \end{bmatrix} \qquad m_0 = 5.871\,\mathrm{mm}$$

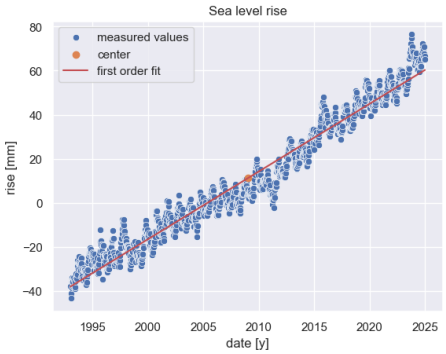
Berechnet man nun den Schwerpunkt der Daten bemerkt man, dass die Gerade $l = x_1 t + x_0$ durch diesen Schwerpunkt verläuft, wie in Abbildung 3.1a gut zu sehen ist. Auf eine Mathematische Herleitung wird hier verzichtet.

3.1 Relativ zur ältesten Messung

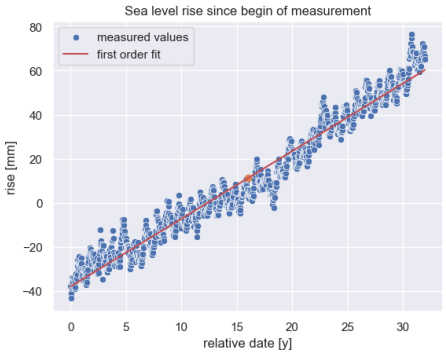
Es ist auch möglich die Regressionsgerade durch Datenpunkte zu legen, bei denen die $\underline{t}$ Werte relativ zum ältesten Messpunkt sind, also $\underline{t} \to \underline{t} - \min \underline{t}$. Wiederholt man nun das selbe Verfahren erhält man

$$\begin{bmatrix} x_1 \\ x_0 \end{bmatrix} = \begin{bmatrix} 3.073\,\mathrm{mm/y} \\ -38.018\,371\,41\,\mathrm{mm} \end{bmatrix} \qquad m_0 = 5.871\,\mathrm{mm}$$

. Es zeigt sich, dass die Werte für x_1 und m_0 übereinstimmen. Dies ist zu erwarten wenn man bemerkt, dass verschobene Geraden dieselbe Steigung haben, und der "Ort" der gerade die "Genauigkeit" der Regression nicht beeinflussen sollte. Der Wert von x_0 stimmt allerdings nicht überein, das ist Ebenfalls zu erwarten, da x_0 dem $y-$Achsenabschnitt der Gerade entspricht, welcher sich beim Verschieben der Gerade ändert. Die Regression durch die modifizierten Datenpunkte ist in Abbildung 3.1b zu sehen.



(a) Regressionsgerade durch die Messpunkte



(b) Regressionsgerade durch die Messpunkte relativ zur ältesten Messung

4 Beliebiges Polynom (n=k)

Beispielhaft wird hier der Fall $n = 6$ betrachtet. Das Resultat ist in Abbildung 4.1 zu sehen. Berechnet man m_0 mit den Residuen erhält man

$$m_0 = 4.947\,\text{mm}$$

Man kann m_0 alternativ auch mit der Formel

$$m_0^2 = \frac{\underline{l}^\mathsf{T}\underline{l} - \underline{l}^\mathsf{T}A^\mathsf{T}\underline{t}}{n-u}$$

berechnen. Man findet, dass in unserem Beispiel die Werte übereinstimmen!

4.1 Kurze Beurteilung des Modells

Da die a-priori Gewichtseinheit σ_0 unbekannt ist, ist es schwierig konkrete Aussagen über das Modell zu machen. Eine Qualitative Analyse von Abbildung 4.1 zeigt allerdings

- dass der Allgemeine Trend der Daten vom Modell gut aufgenommen wird.
- dass Änderungen welche in kurzen Zeiträumen passieren ($\sim$ 1 Jahr) nicht modelliert werden.

Basierend auf dieser Analyse könnte man das Modell mit folgenden Massnahmen verbessern:

- Man könnte die Daten Glätten um diese schnellen Änderungen des Meeresspiegels zu vernachlässigen. Dies ist mit einem Moving-Avarage / Low-pass Filter möglich.
- Man könnte das Modell anpassen z.B. mit einem zusätzlichen Term $x_{n+1}\sin(\omega t)$ Wobei $\omega = 2\pi/\text{Jahr}$ um diese Schnellen Änderungen mit zu modellieren.

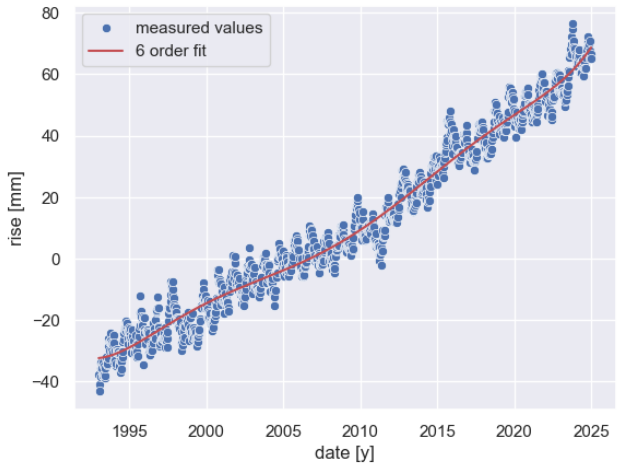


Abbildung 4.1: Linearer Ausgleich mit Polynom 6-ter Ordnung

4.2 Direkte Berechnung der Normalgleichungsmatrix

Es ist möglich die Matrix $N = A^\mathsf{T}A$ welche in der Normalengleichung für die Berechnung der Parameter benötigt ist direkt zu bestimmen, ohne die Design Matrix aufzustellen. Diese Methode ist speicher effizienter, da es vermieden wird die $n \times u$ - grossen Design Matrizen zu verwenden, und direkt die $u \times u$ grosse Normalgleichungsmatrix verwendet wird. Die einträge der Normalgleichungsmatrix sind von der form

$$N_{ij} = (A^\mathsf{T}A)_{ij} = \sum_{k=1}^{N} l_k^i l_k^j = \sum_{k=1}^{N} l_k^{i+j} \qquad i,j \in \{0,\ldots,u\}$$

Vertauscht man nun die Reihenfolge der Einträge um mit der erwarteten Reihenfolge von `np.polyval()` übereinzustimmen erhällt man folgende Python implementation:

```
b = np.ndarray(degree+1)
for i in range(degree+1):
    b[degree-i] = np.sum(y * x**i)

#precompute sums of powers since the Matrix N reuses them
N_entries = np.ndarray(degree*2+1)
for i in range(degree*2+1):
    N_entries[i] = np.sum(x**i)

N = np.ndarray((degree+1, degree+1))

for i in range(degree+1):
    for j in range(degree+1):
        N[degree-i,degree-j] = np.sum(N_entries[i+j])
```

4.2.1 Empirischer Vergleich der Direkten und indirekten Methode

Es ist nun möglich die Laufzeit der beiden Methoden zu vergleichen, in diesem Fall wurden 500′000 zufällige werte zwischen 0 und 1 für $\underline{l}$ und $\underline{t}$ verwendet. Der Grad des Polynoms wurde als $n = 10$ gewählt. Die Resultate sind in Tabelle 4.1 zu sehen.

Methode	Laufzeit
Direkt	(430.0 ± 38.2) ms
Mit Design Matrix	(282 ± 20) ms

Tabelle 4.1: Messungen der Laufzeit der beiden Methoden

Erstaunlicherweise ist die Direkte Methode deutlich langsamer als die Verwendung der Design Matrix. Der Grund dafür ist wahrscheinlich, dass die direkte Berechnung langsame Python-Schleifen verwendet, während die Matrixmultiplikation von `numpy` Sehr schnelle `SIMD` Operationen und Cache optimierte Datenstrukturen verwendet. Nichtsdestotrotz kann es trotzdem Sinn machen die Direkte Methode zu verwenden, falls die Anzahl Datenpunkte so gross ist, dass der Computer auf dem die Berechnungen stattfinden, nicht genügend speicher hat um die Designmatrizen aufzustellen.