# Praktikum 2: Grundlegende Konzepte I

In diesem Praktikum implementieren wir einige der erlernten Methoden für das Beispiel $\sin(2\pi x)$. Konkret werden wir verrauschte Daten aus $\sin(2\pi x)$ generieren, die Maximum-Likelihood-Methode (ML) implementieren und das Model auf den Trainigs- und Testfehlern evaluieren.

**Übung 1.** In dieser Übung generieren wir eigene Daten mit künstlichem Rauschen. Wir implementieren dann die Maximum-Likelihood-Methode (ML) um ein Polynom durch die Trainigsdaten zu fitten. Arbeiten Sie mit der Datei `Ex_1_Polynomial_Regression_ML_skeleton.py` und vervollständigen Sie das Script an der Stelle, welche mit `#Code here` markiert ist.

(a) Vervollständigen Sie die Funktion `generate_noisy_data`, welche Datenpunkte von $\sin(2\pi x)$ generiert und mit einem Rauschen versieht. Die Funktion erwartet `sample_size` und die Variable `sigma` als Input-Parameter. `sigma` kontrolliert dabei das Rauschen. Hier noch einige Hinweise:

- Wir gehen davon aus, dass die $\boldsymbol{x}$-Werte unserer Datenpunkte gleichmässig auf dem Interval $[0, 1]$ verteilt sind. Benutzen Sie `scipy.stats.uniform` und deren Methode `rvs()`, um die $\boldsymbol{x}$-Werte zu generieren.
- Wir nehmen an, dass das Rauschen um den wahren $\boldsymbol{y}$-Wert ($\sigma =$ `sigma`) $\mathcal{N}_{0,\sigma}$ verteilt ist. Benutzen Sie `scipy.stats.norm` und deren Methode `rvs()`, um das Rauschen zu generieren.
- Um die Ziel-Werte $\boldsymbol{t}$ zu generieren, können Sie einfach die wahren $\boldsymbol{y}$-Werte und das Rauschen addieren: $\boldsymbol{t} = \sin(2\pi x) + \text{noise}$.

(b) Vervollständigen Sie die Funktion `plot_noisy_data`, welche Daten als Input erwartet und diese neben der wahren Funktion $\sin(2\pi x)$ plottet. Plotten Sie dann 20 Datenpunkte mit dem Rausch-Parameter $\sigma = 0.1$.

(c) Vervollständigen Sie die Funktion `ml_polynomial_regression`, welche Trainigs-Daten und einen Grad als Input-Parameter erwartet und den Koeffizienten $\boldsymbol{w_{ML}}$ des Regressions-Polynoms mit dem definierten Grad und die erlernte Varianz $\beta_{ML}^{-1}$ zurückgiebt. Benutzen Sie die Maximum-Likelihood-Methode um $\boldsymbol{w_{ML}}$ zu lernen. Hier noch einige Hinweise:

- Wir benötigen die **design Matrix** $X$ für $N$ Datenpunkte und einem Model vom Grad $M$

$$X = \begin{pmatrix} 1 & x_1 & x_1^2 & \dots & x_1^M \\ 1 & x_2 & x_2^2 & \dots & x_2^M \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_N & x_N^2 & \dots & x_N^M \end{pmatrix}$$

Die design Matrix hat die Form einer Vandermonde Matrix Sie können `numpy.vander` nutzen, um diese zu berechnen.

- Die ML Lösung ist definiert als $\boldsymbol{w_{ML}} = (X^T X)^{-1} X^T \boldsymbol{t}$. Um die Lösung numerisch zu berechnen, sollten Sie das zugrundeliegende Least Squares Problem mit `numpy.linalg.lstsq` (womit ein Least Squares Problem der Form $X \cdot \boldsymbol{w} = \boldsymbol{t}$ gelöst werden kann) lösen.

(d) Vervollständigen Sie die Funktion `plot_ml_polynomial_regression`, welche Trainigsdaten und einen Grad als Input-Parameter erwartet und dann die wahre Funktion $\sin(2\pi x)$, die Trainigsdaten selbst und das erlernte Regressions-Polynom mit dem definierten Grad plottet.
*Hinweis:* Um die Polynom-Werte $y(x) = \sum_{k=0}^{M} w_k x^k$ gleichzeitig für alle $x$ zu berechnen, sollten Sie überprüfen, dass folgende Gleichung stimmt ($X$ ist die design Matrix): $\boldsymbol{y} = X \cdot \boldsymbol{w_{ML}}$

(e) Generieren Sie ein Trainigs-Set mit 10 Punkten. Wählen Sie den Grad = 3 und benutzen Sie die Funktion weiter oben, um Ihre Lösung zu plotten. Variieren Sie die Trainis-Set Grösse und den Grad des Polynoms. Vergleichen Sie ausserdem die erlernte Varianz $\beta_{ML}^{-1}$ mit der gewählten Varianz $\sigma^2$ des Rauschens.

**Übung 2.** Nun werden wir das Problem des Overfittings untersuchen. Beim Fitting an unsere Daten haben wir gesehen, dass ein höherergradiges Polynom zu kleineren Fehlern für die Trainigsdaten führt. Beim Evaluieren des Models mittels Testdaten, sahen wir allderings, dass ein höherer Grad nicht umbedingt zu besseren klineren Fehlern führt (die Fehler können sogar grösser werden). Dieses Phänomen nennt man Overfitting.

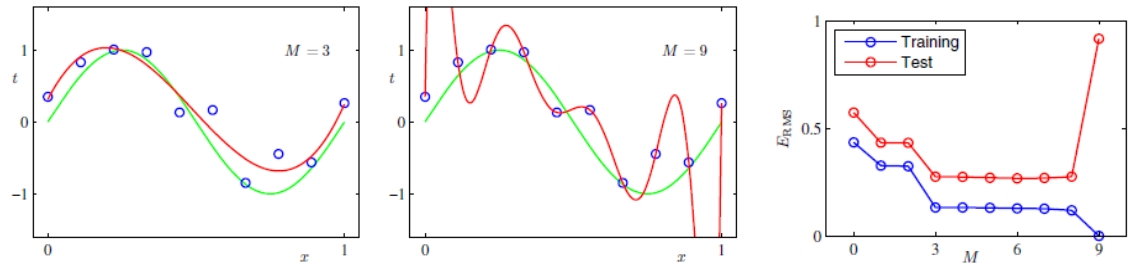


Abbildung 1: $M$ ist der Grad des Polynoms; Grad $M = 9$ führt zu Overfitting auf den Trainigsdaten

Wir werden dieses Phänomen jetzt mit unseren eigenen Daten reproduzieren. Zuerst plotten wir die Regressions-Polynome verschiedener Grade für gegebene Trainigs-Daten. Danach berechnen wir die Trainigs- und Testfehler für Polynome unterschiedlicher Grade, um die "U-Form" in dem Testfehler beobachten zu können. Arbeiten Sie mit der Datei `Ex_2_Training_Test_Error_skeleton.py` und vervollständigen Sie das Script an der Stelle, welche mit `#Code here` gekennzeichnet ist.

(a) Vervollständigen Sie die Funktion `plot_regression_polynomial`, welche Trainigsdaten und eine Liste von Graden als Input-Parameter erwartet. Für jeden

Grad sollten die wahre Funktion $\sin(2\pi x)$, die Trainigsdaten und das Regressions-Polynom geplottet werden. Wenden Sie die Funktion für ein Trainigs-Set mit 10 Punkten und Polynomen von Grad 1, 3 und 9 an.

(b) Vervollständigen Sie die Funktion `rmse`, welche den Root Mean Square Error (RMSE)

$$E_{RMS}(D) = \sqrt{\frac{\sum_{k=1}^{N}(t_k - y(x_k))^2}{N}}$$

berechnet, wobei $D = \{(x_k, t_k)|1 \leq k \leq N\}$ das Trainigs-Set und $y(x)$ das Regressions-Polynom darstellen.

(c) Vervollständigen sie die Funktion `plot_errors`, welche ein Set von Trainigs-Daten, Test-Daten und einem Maximalen Grad als Input-Parameter erwartet. Sie sollte für jeden Grad $1 \leq M \leq$ `max_degree` den Trainigs- und den Test-Fehler berechnen und diese dann in einem Grad-Fehler-Diagramm plotten. Führen sie die Funktion mit einem Set von 20 Punkten, einem Test-Set von mindestens 20000 Punkten und einem maximalen Grad von 20 aus.