Universität Osnabrück

Seminar Lernen auf Datenströmen
Wintersemester 14/15

Prof. Werner Brockmann

Recursive Least M-estimate
Implementation und Performance

Autor:
Christian Heiden
cheiden@uos.de

Betreuer:
Jan H. Schoenke
Jonas Schneider

28. Januar 2015

Inhaltsverzeichnis

1 Einleitung

Wenn die Eingangssignale eines adaptiven Filters mit Impulsen versetzt sind kann dies zu einem verzerrten Adaptionsverhalten führen. Hierbei stellen Impulse bzw. Ausreißer Datenpunkte dar, welche nicht in den Lernprozess miteinbezogen werden sollen. Die Entstehung solcher Impulse variiert je nach Anwendung. Beispielsweise können die Daten durch eine Signalübertragung beschädigt worden sein und daher Werte aufweisen, welche das Lernverhalten verfälschen können.

Ein Lernverfahren, welches mit impulsversetzten Daten umgehen kann, ist das adaptive Filter *Recursive Least M-estimate (RLM)* [1]. Ziel dieser Ausarbeitung ist das Vorstellen und Bewerten dieses Verfahrens.

Zu Beginn wird in Abschnitt (2) der RLM vorgestellt. In Abschnitt (3) wird auf die Performance des RLM in verschiedenen Szenarien eingegangen und jeweils bewertet. Zum Schluss wird in Abschnitt (4) diese Ausarbeitung zusammengefasst.

2 Recursive Least M-Estimate (RLM)

Zunächst wird in Abschnitt (2.1) eine Einführung in die Arbeitsprozesse robuster Lernverfahren gegeben. Anschließend wird das Lernverfahren des RLM erläutert. Dazu wird zunächst kurz die Funktionsweise des *Recursive Least Squares (RLS)* Algorithmus beschrieben (Abschnitt (2.2)), da dieser die Grundlage des RLM darstellt. Anschließend wird auf die Unterschiede beider Lernverfahren eingegangen und die Funktionsweise des RLM im Detail beschrieben (Abschnitt (2.3)).

2.1 Robuste Lernverfahren

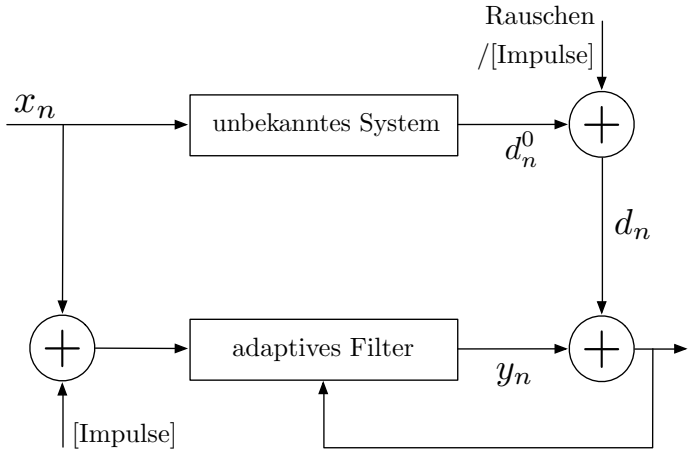


Abbildung 1: Struktur eines Lernprozesses (Vergleiche Quelle: [1]).

In Abb. (1) ist die Struktur des Lernprozesses eines robusten adaptiven Filters zu sehen. Ziel des Lernverfahrens ist die Annäherung seines Outputs an den des unbekannten Systems (US). Das US steht für die abstrakte Betrachtung der Zielfunktion, die approximiert werden soll. Sei x_n das Lerndatum zum Zeitpunkt n, welches den Input des adaptiven Filters darstellt, so soll der Output des Filters y_n möglichst nah am Output des US d_n^0 liegen. Als Herausforderung stellt sich heraus, dass das Datum welches dem Lernverfahren für den Vergleich zur Verfügung steht mit Rauschen versetzt sein kann. Folglich findet die Annäherung von y_n an $d_n = \mathit{Rauschen} + d_n^0$ statt. Als *Rauschen* wird im Rahmen dieser

Ausarbeitung eine Gaussverteilte Zufallsvariable angenommen. Nachdem das Filter seinen Output y_n generiert hat wird der Fehler e_n zu d_n berechnet und es werden die Parameter des Filters in Abhängigkeit von e_n angepasst.

Eine weitere Herausforderung sind impulsversetzte Daten. Dabei kann x_n oder d_n mit Impulsen versetzt sein. Ein Impuls lässt sich als Offset betrachten, welches auf die entsprechenden Werte addiert wird. Robuste Lernverfahren haben als Aufgabe diese Impulse zu erkennen und ihre Wirkung auf die Adaption zu hemmen.

Im dieser Ausarbeitung wird sich implizit auf impulsversetzte d_n^0-Werte bezogen, da die Implementation der Impulse in Matlab so durchgeführt wurde.

2.2 RLS

Der *Recursive Least Squares (RLS)* Algorithmus [2, p. 195ff] ist ein Lernverfahren 2. Ordnung, welches den quadratischen Fehler auf allen gegebenen Daten minimiert. In den Gleichungen (1) und (2) sind allgemeine Formeln für Lernverfahren 2. Ordnung zum Update von α_n und S_n zu sehen.

$$\alpha_n = a \cdot \alpha_{n-1} + b \cdot (y_n - \tilde{y}_n)) \cdot S_{n-1}\varphi(x_n) \tag{1}$$

$$S_n = c \cdot S_{n-1} + d \cdot S_{n-1}\varphi(x_n)\varphi(x_n)^T S_{n-1} \tag{2}$$

α_n stellt den Parametervektor und S_n die Korrelationsmatrix des Lernverfahrens dar. y_n ist der gewünschte und $\tilde{y}_n$ der berechnete Output. Es gilt $\tilde{y}_n = \alpha_n\varphi(x_n)$ wobei $\varphi(x_n)$ die Anwendung des Inputs x_n auf die verwendeten Basisfunktionen ausdrückt.

Die Festlegung der Koeffizienten a, b, c, d identifiziert ein solches Lernverfahren eindeutig. Im Falle des RLS sind die Belegungen der Koeffizienten in den Gleichungen (3) und (4) zu sehen.

$$a = c = 1 \tag{3}$$

$$b = d = \frac{1}{\lambda + \varphi(x_n)^T S_{n-1}\varphi(x_n)} \tag{4}$$

$0 \ll \lambda < 1$ wird auch als *Vergessenheitsfaktor* bezeichnet. Aufgrund seines antiproportionalen Einflusses in b und d trägt ein hoher Wert zu einer stärkeren Vernachlässigung neuer Werte auf die Änderung von α_n und S_n bei.

2.3 RLM

Der *Recursive Least M-estimate* (*RLM*) Algorithmus [1] ist ein robuster Lernalgorithmus, welcher auf Basis des RLS arbeitet. Es soll zunächst grob auf die mathematische Herleitung des Algorithmus eingegangen werden bevor im Anschluss die Eigenschaften des RLM diskutiert werden.

Die Loss-Funktion des RLM ist gegeben als:

$$L_n = \sum_{i=0}^{n} \lambda^{n-i}\rho(e_i)$$

Hierbei sei e_n definiert als der aktuelle Fehler $e_n = y_n - \tilde{y}_n$. Die abschnittweise definierte Funktion $\rho(e_n)$

$$\rho(e) = \begin{cases} \frac{e^2}{2} & 0 < |e| < \xi, \\ \xi|e| - \frac{\xi^2}{2} & \xi \leq |e| < \Delta_1, \\ \frac{\xi^2}{2}(\Delta_2 + \Delta_1) - \frac{\xi^2}{2} + \frac{\xi}{2}\frac{(|e|-\Delta_2)^2}{\Delta_1 - \Delta_2} & \Delta_1 \leq |e| < \Delta_2, \\ \frac{\xi}{2}(\Delta_2 + \Delta_1) - \frac{\xi^2}{2} & \Delta_2 \leq |e| \end{cases}$$

stellt den *M-Estimate*, bzw. *M-Schätzer* dar und bildet vom aktuellen Fehler e_n auf einen positiven reellen Wert ab. Auf die Eigenschaften von $\rho(e_n)$, insbesondere ihre Grenzen ξ, Δ_1 und Δ_2, wird später eingegangen. Die Wahl der Loss-Funktion L_n ist ein entscheidender Punkt in der Unterscheidung mit anderen Lernverfahren. Beispielsweise verwendet der RLS als Loss-Funktion den quadrierten aktuellen Fehler, und der PRRLS den quadrierten gewichteten Fehler (siehe Abschnitt 3.7.1). Das Ziel eines Lernverfahren ist mittels Adaption seiner α_n- und S_n- Werte die Vorhersagefehler e_{n+1} basierend auf der Loss-Funktion zu minimieren.

Dazu wird die Ableitung von L_n mit 0 gleichgesetzt, woraus sich die Gleichung

$$R_n \alpha_n = P_n \tag{5}$$

mit

$$\begin{aligned} R_n &= \lambda R_{n-1} + q(e_n)\varphi(x_n)\varphi(x_n)^T \\ P_n &= \lambda P_{n-1} + q(e_n) y_n \varphi(x_n) \end{aligned} \tag{6}$$

ergibt, wobei $R_n = S_n^{-1}$ als *Korrelationsmatrix* und P_n als *Kreuzkorrelationsvektor* bezeichnet werden. Um an das Adaptionsupdate S_n für den RLM zu kommen muss folglich R_n nun invertiert werden. Dies ist möglich durch Anwenden des Matrix Inversions Lemma (Lemma aus [1] übernommen)

$$(A + \mu x y^T)^{-1} = A^{-1}\{I - (\mu x y^T A^{-1})/(1 + \mu y^T A^{-1} x)\}$$

auf Gleichung (6) mit $A = \lambda R_{n-1}$, $x = y = \varphi(x_n)$ und $\mu = q(e_n)$. So ergibt sich für das Adaptionsupdate

$$S_n = \frac{1}{\lambda}(I - d \cdot S_{n-1}\varphi(x_n)\varphi(x_n)^T)S_{n-1} \tag{7}$$

mit

$$b = d = \frac{q(e_n)}{\lambda + q(e_n) \cdot \varphi(x_n)^T S_{n-1}\varphi(x_n)} \tag{8}$$

Durch die Gleichungen (5) und (7) ergibt sich das Update des α_n-Vektors zu den Formeln in (1) und (3).

Der RLM unterscheidet sich vom RLS größtenteils in den Variablen b und d. Denn verglichen mit Gleichung (4) ist lediglich ein Faktor

$$q(e_n) = \frac{\rho'(e_n)}{e_n}$$

in Zähler und Nenner hinzugekommen.

Wird $\rho(e_n)$ abgeleitet und an e_n normalisiert, so ergibt sich $q(e_n)$ zu einer Funktion, welche vom aktuellen Fehler e_n auf das Intervall $[0, 1]$ abbildet. Dies ist ebenfalls in Abb. (2) zu erkennen.

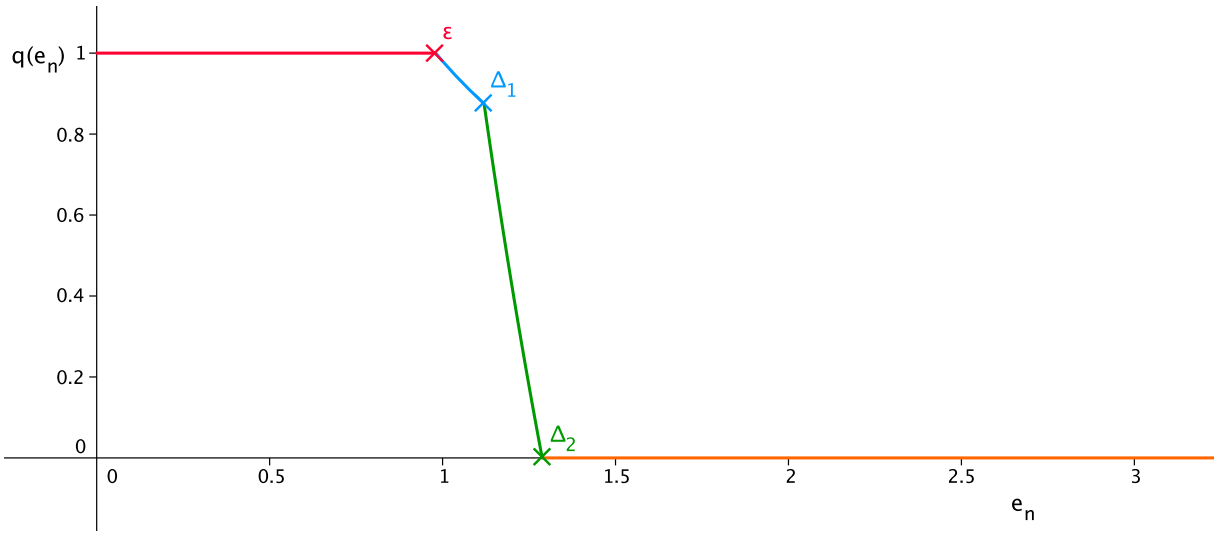


Abbildung 2: Graph zu $q(e_n)$ bei einer Standardabweichung von $\sigma_n = 0.5$.

Wenn $q(e_n) = 1$ gilt, verhält sich der RLM genau wie der RLS, da die Gleichungen (4) und (8) dann gleich sind. Für alle Werte $0 < q(e_n) < 1$ ergeben sich beim RLM kleinere Werte für b und d als beim RLS. Sollte $q(e_n) = 0$ gelten, so wird der aktuelle Fehler nicht in das Update von α_n und S_n miteinbezogen, da dann $b = d = 0$ gilt (vergleiche dazu Formeln (1) und (2)). Kleine q-Werte sorgen also für eine Hemmung des aktuellen Fehlers auf die Updates von α_n und S_n. In anderen Worten fomuliert: große Fehler beeinflussen das Update von α_n und S_n weniger, als kleine Fehler.

Als Vorrausetzung gilt folglich, dass Impulse bzw. Ausreißer das Lernverhalten nicht beeinflussen sollen. Diese Annahme muss vom jeweiligen Anwendungsszenario abhängig gemacht werden, da es ggf. wünschenswert ist auf Ausreißer Rücksicht zu nehmen. In solchen Szenarien ist von der Verwendung des RLM abzuraten (für eine detallierte Diskussion siehe Abschnitt (3.7)).

Maßgeblich für die Entscheidung ob der aktuelle Fehler als Impuls aufgefasst werden kann oder nicht sind die Grenzen ξ, Δ_1 und Δ_2 von $\rho(e_n)$. Wie in Abb. (2) zu sehen, bedeutet ein Fehler kleiner als ξ, dass sich der Algorithmus wie der RLS verhält, wohingegen ein Fehler größer als Δ_2 dafür sorgt, dass beim Update von α_n und S_n keine Rücksicht auf den Fehler genommen wird. Die Grenzen müssen so gesetzt werden, dass mit ihnen Impulse von normalen Datensätzen unterschieden werden können. Beim RLM-Algorithmus werden die Grenzen in jeder Iteration neu festgelegt. Als Basis dient die Standardabweichung der letzten N Datensätze. Es werden nachfolgende Eigenschaften festgelegt, die besagen wieviel Prozent der Daten von den jeweiligen Grenzen miteinbegriffen werden sollen:

$$\begin{aligned} 95\% &\quad \text{der Daten} \leq \xi \\ 5\% &\quad \text{der Daten} > \xi \\ 2.5\% &\quad \text{der Daten} > \Delta_1 \\ 1\% &\quad \text{der Daten} > \Delta_2 \end{aligned}$$

Daraus können die folgenden direkten Abhängigkeiten der Grenzen zur Standardabweichung σ_n getroffen werden:

$$\begin{aligned} \xi &= 1.96 \cdot \sigma_n \\ \Delta_1 &= 2.24 \cdot \sigma_n \\ \Delta_2 &= 2.576 \cdot \sigma_n \end{aligned} \tag{9}$$

Die Varianz wird mit folgender Formel errechnet:

$$\hat{\sigma}_n^2 = \lambda_\sigma \hat{\sigma}_{n-1}^2 + C_1(1-\lambda_\sigma) \cdot \text{median}(A_n)) \qquad \text{mit} \quad C_1 = 1.483(1 + \frac{5}{N-1})$$

$A_n = \{e_n^2, \ldots, e_{n-N+1}^2\}$ ist eine Menge der vergangenen N quadrierten Fehler, λ_σ ist ein zusätzlicher Vergessenheitsfaktor und C_1 eine Konstante um die Werte für die Berechnung von $\hat{\sigma}_n^2$ anzupassen. Die aktuelle Varianz $\hat{\sigma}_n^2$ errechnet sich grob betrachtet aus der vorherigen Varianz $\hat{\sigma}_{n-1}^2$ und dem Median der letzten N Fehler. Durch die Verwendung des Medians gehen *vereinzelte* Ausreißer folglich nicht in die Berechnung der aktuellen Varianz mit ein. Dadurch soll erzielt werden, dass die Varianz lediglich das Rauschen erfasst und somit ξ, Δ_1 und Δ_2 Impulsfrei festgelegt werden können. Es ist anzumerken, dass auf Grenzen/Probleme der beschriebenen Berechnungen in Abschnitt (3) näher eingegangen wird.

2.3.1 Ablauf und Implementation

Dieser Abschnitt behandelt den eigentlichen Ablauf des RLM. Verknüpfend wird hier Bezug auf dessen Implementation in Matlab genommen. Der Algorithmus ist als Bestandteil der *UOSLib* [3] geschrieben. In der Datei `icl_initILS_RLM.m` (siehe Anh. (B.1)) werden Initialisierungen des RLM vorgenommen und in `icl_learn_RLM.m` (siehe Anh. (B.2)) befindet sich der Code, welcher in jeder Iteration des Lernprozesses ausgeführt wird.

In Tab. (1) sind alle vorgestellten Parameter des RLM aufgeführt samt deren Entsprechungen im Code und ihrer default-Werte.

Name	**Symbol**	**Code**	**Defaultwert**
Vergessenheitsfaktor	λ	`ILS.g`	1
Korrelationsmatrix	S_0	`ILS.S`	10^5`*eye(length(`x_0`))`
# Fehler im Median	N	`ILS.errhist`	30
Standardabweichung	σ_0	`ILS.dev`	1
Vergessenheitsfaktor Median	λ_σ	`ILS.gdev`	0.9

Tabelle 1: Parameter des RLM samt deren Entsprechungen im Matlab-Code und ihrer default-Werte.

In der Initialisierungsphase werden zunächst alle Parameter gesetzt. Die Wahl der Parameter kann explizit durch den Anwender geschehen, oder wird, sofern dieser keine Werte festlegt, mit default-Werten getan. Anschließend werden die Grenzen ξ ($\widehat{=}$`ILS.eps`), Δ_1 ($\widehat{=}$`ILS.delta_1`) und Δ_2 ($\widehat{=}$`ILS.delta_2`) entsprechend der Standardabweichung initialisiert.

Der Ablauf des Lernprozesses in einer Iteration lässt sich wie folgt zusammenfassen:

1. Aufruf von `icl_learn_RLM` mit aktueller Parameterkonfiguration in `ILS`, dem Datenpunkt `x`, dem vorhergesagten Wert `yp` und echten Wert `y`
2. Der aktuelle Fehler `e_n` wird berechnet
3. `x_n` wird auf Basisfunktionen angewendet
4. $q(e_n)$ wird mittels der Funktion `m_estimate(e_n, ILS)` basierend auf Fehler `e_n` und aktuellen Grenzen in `ILS` angewendet und Wert in `q_n` abgelegt
5. Der `gain`-Vektor wird berechnet
6. Der α_n Vektor wird aktualisiert
7. Die Adaptionsmatrix S_n wird aktualisiert

8. Die neue Standardabweichung σ_{n+1} wird berechnet und `ILS.eps`, `ILS.delta_1` und `ILS.delta_2` werden neu gesetzt.

Bei den Updates von α_n und S_n weicht der Code von der in der Ausarbeitung verwendeten Formeln ein wenig ab, verändern das Resultat jedoch nicht. Anstelle der in den Gleichungen (1) und (7) verwendeten Variablen b und d wird in der Implementation der gain-Vektor berechnet und in den Update-Formeln verwendet (vergleiche dazu mit den Zeilen 41 und 44 in in Anh. (B.2)). Der Code orientiert sich stärker an der Herangehensweies von [1], während diese Ausarbeitung mit Hinsicht auf Konsistenz mit den Vorbereitungsfolien geschrieben wurde.

Zur Performance-Evaluierung von impulsversetzten Daten (siehe Abschnitt (3.6)) existiert eine weitere Funktion, welche in `icl_base` aufgerufen wird um Offsets auf bestimmte Datensätze zu addieren. Die Methode `icl_genImpulses` (siehe Anh. (B.3)) nimmt als Parameter das Datenset und addiert Offsets, welche als weitere Parameter der Funktion übergeben werden können. Es sind nicht alle Funktionalitäten fertiggestellt, da die option `impulseArray` für die Evaluierung ausgereicht hat.

3 Performance

In diesem Abschnitt wird auf die Performance des RLM in verschiedenen Szenarien eingegangen. Jedes Szenario wird zum Vergleich sowohl mit dem RLM als auch mit dem RLS durchlaufen. Dies soll helfen den RLM besser vom RLS abgrenzen zu können und ihn somit leichter in einen Kontext einordnen zu können.

Die verwendeten Maße zur Messung der Performance sind

$$e_c(T) = \sum_{t=1}^{T} L(f(x_t, \alpha_t), y_t) \qquad \text{Kumulativer Loss}$$

$$e_d(T) = \sum_{t=1}^{T} L(f(x_t, \alpha_T), y_t) \qquad \text{Loss auf allen Daten}$$

$$e_g(T) = \sum_{x \in X} L(f(x, \alpha_T), y_t) \qquad \text{Loss auf Zielfunktion}$$

Sofern nicht anders in dem jeweiligen Anwendungsfall angegeben, ist das Standardsetup des RLM und des RLS wie in Tab. (2) angegeben.

Parameter	**RLM**	**RLS**
S	10000	10000
λ	0.99	0.99
λ_σ	0.85	-
N	15	-
σ_0	1	-

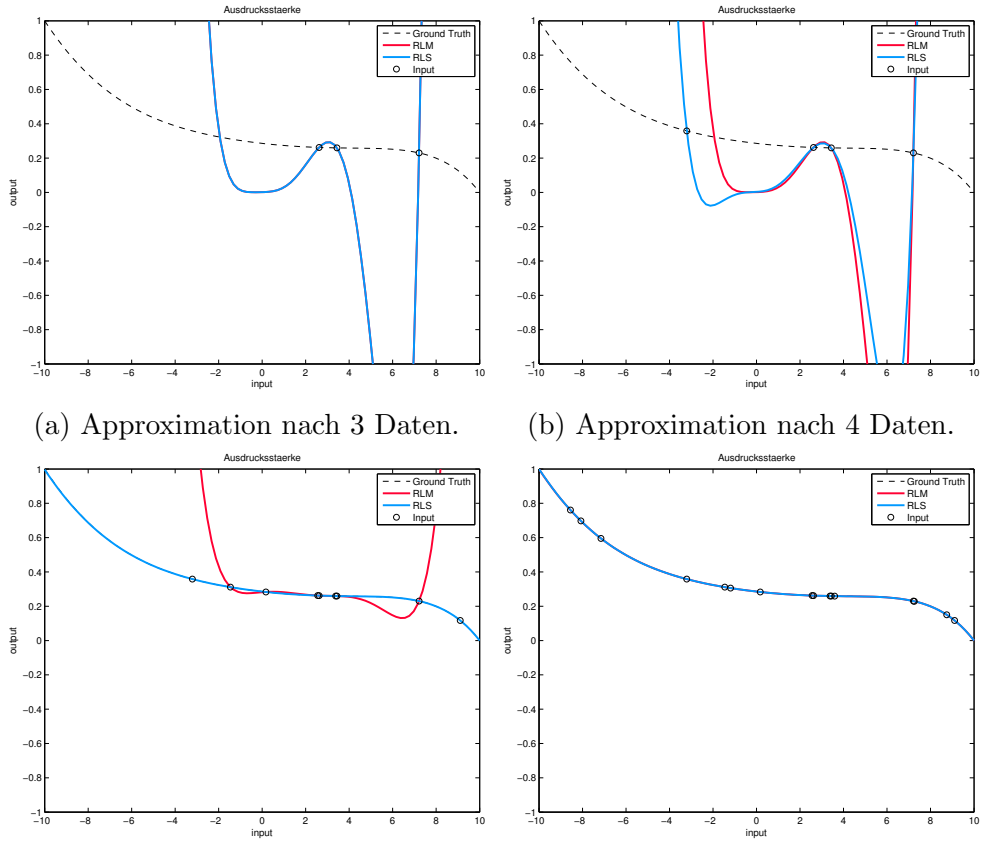
Tabelle 2: Default-Setup des RLM/RLS.

Zu Beginn wird in Abschnitt (3.1) auf Szenarien mit Approximatoren verschiedener Ausdrucksstärken eingegangen. Dort soll auch eine Einführung in das allgemeine Approximationsverhalten des RLM gegeben werden. Anschließend werden in Teil (3.2) Ergebnisse vorgestellt, welche den Umgang des RLM mit verrauschten Daten beschreiben. In Abschnitt (3.3) wird die Wirkung von Trajektorieneffekten in den Daten erörtert. Danach

werden zeitvariante Zielfunktionen betrachtet (Abschnitt (3.4)). Im Anschluss wird auf nichtlineare Zielfunktionen eingegangen (Abschnitt (3.5)). Beim letzten Szenario in Abschnitt (3.6) wird auf die Ergebnisse des RLM bei impulsversetzten Daten eingegangen. Zum Schluss werden die erarbeiteten Ergebnisse in Beziehung gebracht und bewertet und mit einem anderen robusten Lernverfahren, dem PRRLS, verglichen.

3.1 Ausdrucksstärke

Als Setup[1] wird eine polynomielle Zielfunktion mit einem Polynom 6. Grades approximiert. In den Abb. (3a), (3b), (3c) und (3d) sind die Approximationen von RLM und RLS nach 3, 4, 9 und 16 Lerndaten zu sehen.



(a) Approximation nach 3 Daten.

(b) Approximation nach 4 Daten.

(c) Approximation nach 9 Daten.

(d) Approximation nach 16 Daten.

Abbildung 3

Bis zum dritten Lerndatum verhalten sich beide Lernverfahren gleich, was in Abb. (3a) an den übereinanderliegenden Graphen erkennbar ist. Beim vierten Lerndatum (siehe Abb. (3b)) ist zu sehen, dass sich beide Lernverfahren nun unterschiedlich verhalten. Während der RLS das präsentierte Lerndatum abbildet, zeigt der RLM keine Veränderung gegenüber Schritt 3. Der Grund für dieses Verhalten lässt sich anschaulich an Abb. (3b) zeigen. Der Fehler e_n zwischen der approximierten Kurve des RLM und des neuen Lerndatums ist „hoch“. Bei einem „hohen“ Fehler wird $q(e_n)$ auf 0 gesetzt und hemmt somit das Update von α_n und S_n. Als Resultat zeigt der Graph des RLM in Schritt 4 also dieselbe Approximation wie im Schritt zuvor.

[1] **Setup**: `Zielfunktion=poly, # Datensätze=100, # Groundtruth=100, noise=0.0, minpath=false, Approximator=Polynom Grad 6`

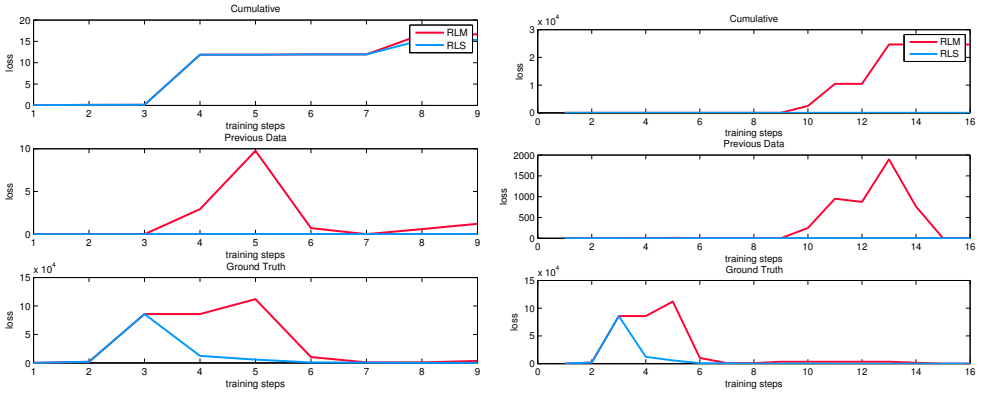
Dieses Verhalten ist stark von den gewählten Parametern abhängig und kann dementsprechend beeinflusst werden. Da die Grenzen ξ, Δ_1 und Δ_2 steuern, ab wann ein Fehler als „hoch“ zu betrachten ist und diese wiederum abhängig von der Standardabweichung sind, kann σ_0 größer gewählt werden, sodass q den auftretenden Fehler in Schritt 4 nicht mit 0 bewertet und somit ein Update von α_n und S_n zulässt.

Eine weitere Möglichkeit ist, dass Updateverhalten der Standardabweichung zu verändern. Mittels dem Vergessenheitsfaktor λ_σ kann bestimmt werden, wie sehr die Standardabweichung neue Fehler mit einschließt. Wird der Wert „klein“ gewählt, so hat der Median der vergangenen Fehlerwerte „größeren“ Einfluss auf σ (siehe Gleichung (9)). Hier ist zu beachten, dass diese Veränderung das Update von σ lediglich weniger träge macht. Es bedeutet nicht, dass jeder Fehler sofort die Standardabweichung beeinflusst, da diese weiterhin abhängig vom Median der vergangenen Fehlerwerte ist.

Es gibt eine weitere Möglichkeit das Approximationsverhalten in dem Szenario in Abb. (3) zu variieren. Eine Änderung in der Anzahl der vergangenen Fehler N über welche der Median gebildet wird, kann dafür sorgen, dass der Median schneller bzw. langsamer auf Fehler reagiert. Denn damit z.B. große Fehler vom Algorithmus beachtet werden sollen, müssen diese mindestens $\lceil \frac{N}{2} \rceil$ der Objekte ausmachen über die der Median gebildet wird. Wenn also N „klein“ gewählt wird, haben wenige Fehler bereits die Möglichkeit Einfluss auf σ zu nehmen.

Ab Lernschritt 9 lässt sich Abb. (3c) erkennen, dass der RLS, im Gegensatz zum RLM, die unterliegende Funktion bereits gut angenähert hat. Der RLM verbleibt weitere 4 Lerndaten in dieser Form, bis zu dem Punkt, wo ein Datum nah genug an seiner Approximation liegt, sodass der Fehler klein genug ist um ein Update zu ermöglichen. Dieses Verhalten wird ebenfalls in den Szenarien der Abschnitte (3.3) und (3.5) besprochen. Schlussendlich liegt der RLM nach dem 16. Lerndatum wieder Deckungsgleich auf dem RLS.

Nun sollen die Performance-Ergebnisse der besprochenen Approximationen betrachtet werden. In Abb. (4b) sind die Werte der einzelnen Performancemaße der 16 Lerndaten zu sehen.



(a) Performance nach 9 Daten. (b) Performance nach 16 Daten.

Abbildung 4

Beim dritten Lerndatum liegen die Kurven beider Lernverfahren in jedem Performancemaß übereinander, wie auch in der Approximation erkennbar ist. Da der Approximator mit Grad 6 in der Lage ist die bereits präsentierten Daten abzubilden zeigt der Loss auf den vergangenen Daten keinen Anstieg. Der Fehler zu den GroundTruth-Daten steigt hier jedoch an. Der Grund für dieses Verhalten ist, dass bisher ausschließlich die 3 Datensätze abbgebildet werden sollten und wie in Abb. (3a) zu erkennen ist, jedoch diese Informationen nicht ausreichen um die GroundTruth-Funktion anzunähern. In Schritt 4

verhalten sich beide Algorithmen unterschiedlich, was auch in den Performance-Werten sichtbar wird (siehe Abb. (4a)). Da der RLM den neuen Datenpunkt nicht abbildet, steigt der Loss auf den vergangenen Datensätzen. Durch die unveränderte Konfiguration seiner Parameter bleibt der Loss auf den GroundTruth-Daten gleich. Dieser Effekt lässt sich ebenfalls ab dem neunten Lerndatum wiederfinden. Wie in Abb. (4b) zu sehen ist steigt der Previous-Data Loss bis zum 13 Lerndatum an. Solange werden Daten präsentiert, bei welchen der Fehler so groß ist, dass der RLM diese als Impulse einschätzt und somit keine Adaption herbeiführt. Erst das 14. Lerndatum liegt nah genug an der Approximation, sodass der RLM eine Adaption vornimmt und somit die vergangenen Daten besser abbildet. Für die restlichen 94 Lerndaten verhalten sich RLM und RLS gleich, da beide die GroundTruth-Funktion ab Lerndatum 16 abbilden.

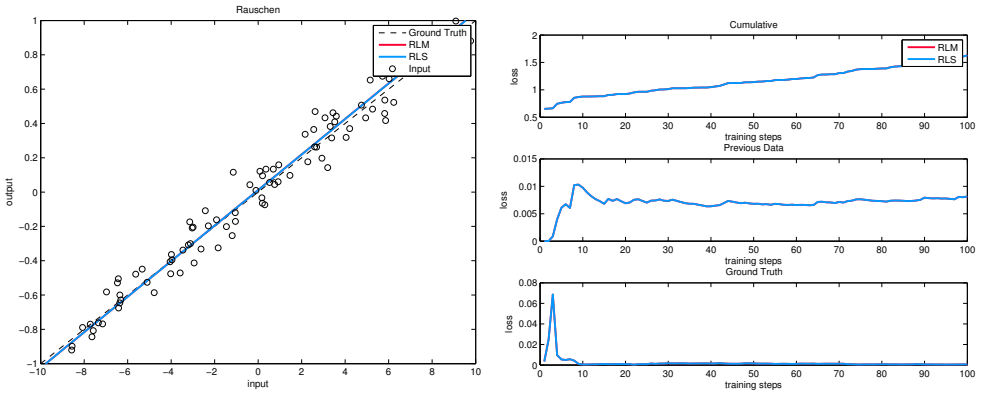
Es wurde ein weiteres Setup[2] getestet, welches einen linearen Approximator auf polynomiellen GroundTruth-Daten anwendet (siehe Anh. (12a) und (12b)). Es ist zu beobachten, dass sich die Performance-Ergebnisse beider Verfahren nicht sehr unterscheiden. Der Grund hierfür ist, dass mittels eines linearen Approximators der Fehler zu den präsentierten Daten kleiner bleibt, als bei einem Polynom höheren Grades. Dadurch wird die Hemmung klein gehalten, wodurch das Verhalten des RLM stark dem des RLS ähnelt.

Zusammenfassend lässt sich sagen, das ein Approximator mit einer hohen Ausdrucksstärke dazu führt, dass die initiale Phase in der der RLS sich auf die Zielfunktion einpendeln muss, verlängert wird, da Lerndaten fälschlicherweise als Ausreißer interpretiert werden. Die Performance nach der initialen Phase verhält sich gleich wie beim RLS.

3.2 Rauschen

In diesem Abschnitt werden RLM und RLS in zwei Setups auf verrauschten Daten angewendet.

Beim ersten Setup[3] wird ein linearer Approximator und eine lineare Zielfunktion verwendet, welche dem Lernverfahren verrauscht präsentiert wird. Wie in den Abb. (5) erkennbar ist, sind Approximierung und Performance des RLM und RLS identisch.



(a) (b)

Abbildung 5: Approximation und Performance-Ergebnisse bei linearem Approximator auf linearen GroundTruth-Daten mit Rauschen.

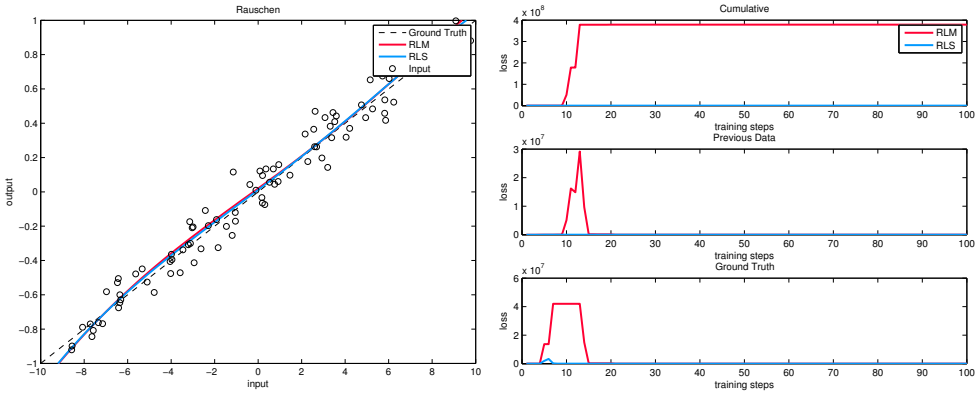
Verrauschte Daten resultieren beim RLM in einer größeren Standardabweichung, so-

[2]**Setup**: `Zielfunktion=poly, # Datensätze=100, # Groundtruth=100, noise=0.0, minpath=false, Approximator=Polynom Grad 1`

[3]**Setup**: `Zielfunktion=linear, # Datensätze=100, # Groundtruth=100, noise=0.1, minpath=false, Approximator=Polynom Grad 1`

dass die auftretenden Fehlerwerte, trotz Rauschen, nicht zu einer Hemmung des Parameterupdates führen.

Das zweite Setup[4] soll diesen Aspekt anhand eines polynomiellen Approximators 5. Grades untersützen und zusätzlich einige Inhalte aus Abschnitt (3.1) erneut verdeutlichen.



(a) (b)

Abbildung 6: Approximation und Performance-Ergebnisse bei polynomiellen Approximator 5. Grades auf linearen GroundTruth-Daten mit Rauschen.

Wie in Abb. (6a) zu sehen ist, haben beide Lernverfahren nach 100 Datensätzen trotz hoher Ausdrucksstärke die Kurve relativ gut angenähert. Mit Hinblick auf die Performance-Ergebnisse in Abb. (6b) lässt sich jedoch ein großer Unterschied feststellen. Der RLM zeigt einen hohen Loss bei allen Performancemaßen im Bereich vor dem 16. Lerndatum. Grund für dieses Verhalten sind weniger die verrauschten Daten, als die hohe Ausdrucksstärke des Approximators. Für die Erklärung dieses Sachverhaltes sei auf das erste Setup in Abschnitt (3.1) verwiesen.

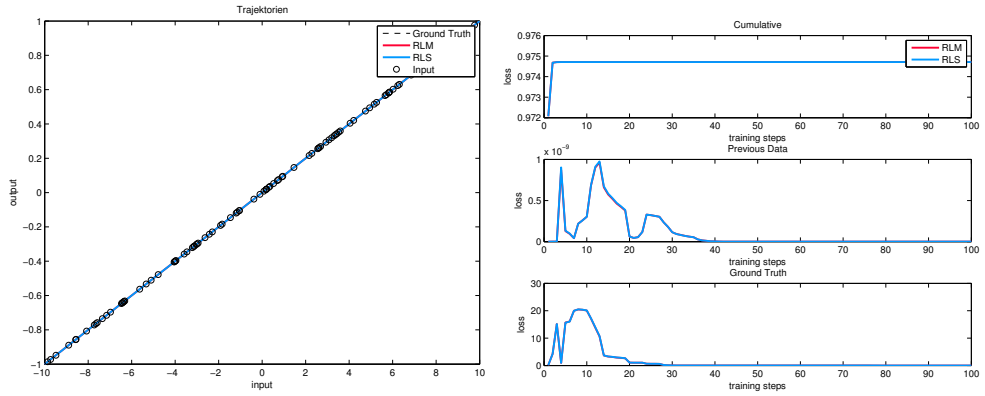
Zusammenfassend lässt sich sagen, dass der RLM mit Rauschen keine Problem hat, solange die verrauschten Daten mit einer gewissen Varianz um die Zielfunktion liegen (z.B. Gauss-verteilt).

3.3 Trajektorieneffekte

In diesem Teil der Ausarbeitung wird auf Trajektorieneffekte bei den Lerndaten eingegangen. Es wird ein Setup[5] vorgestellt in dem eine lineare Zielfunktion mit einem Polynom 5. Grades approximiert wurde. In Abb. (7a) ist zu sehen, dass sich RLM und RLS zu allen Zeitpunkten gleich verhalten.

[4]**Setup**: `Zielfunktion=linear, # Datensätze=100, # Groundtruth=100, noise=0.1, minpath=false, Approximator=Polynom Grad 5`

[5]**Setup**: `Zielfunktion=linear, # Datensätze=100, # Groundtruth=100, noise=0.0, minpath=true, Approximator=Polynom Grad 5`



(a) (b)

Abbildung 7: Approximation und Performance-Ergebnisse bei polynomiellen Approximator 5. Grades auf linearen GroundTruth-Daten mit Trajektorieneffekten.

Die Erklärung für dieses Verhalten liegt der Tatsache zugrunde, dass immer jenes Datum als nächstes präsentiert wird, welches die Eigenschaft hat, die kürzeste Distanz zum vorhergehenden Datum zu haben. Da der RLM, genau wie der RLS, versucht den Fehler zu allen Daten zu minimieren und in diesem Szenario, die Daten immer nahe an den vergangenen Daten liegen, führt dies dazu, dass die Fehlerwerte nur mit einer geringen Wahrscheinlichkeit groß werden. Diese Besonderheit sorgt dafür, dass das Verhalten nicht die gleichen hohen Loss-Werte wie bei dem Szenario aus Abschnitt (3.2) aufweist. Da keine Hemmung des Parameter- und Adaptionsupdates stattfindet, verhalten sich beide Algorithmen gleich.

Bei der Erklärung dieses Sachverhalts gilt die implizite Annahme, dass die Datendichte ausreichend hoch ist bzw. die Daten nicht sehr weit voneinander entfernt liegen. In beiden Fällen kann es passieren, dass die Abstände zum nächsten Lerndatum für einen zu hohen Fehler e_n sorgen, welcher das Update von α_n und S_n hemmt.

Mit Blick auf die Performancemaße in Abb. (7b) lässt sich sehen, dass beide Algorithmen mehr Zeit benötigen die Zielfunktion abzubilden als vergleichbare Szenarien der anderen Abschnitte. Dies liegt an der Tatsache, dass der Approximator anfänglich nur einen Teil der Zielfunktion annähern kann, während die Zielfunktion insgesamt auf einem größeren Intervall definiert ist, dem Algorithmus jedoch schlichtweg nicht bekannt ist zu Beginn.

Auf die Vorstellungen zweier weiterer Setups[6,7] mit kleinerem Approximatorgrad wird hier verzichtet, da die sich Ergebnisse analog zu dem erstbeschriebenen Setup verhalten. Die Ergebnisse sind im Anhang in den Abb. (13) und (14) zu sehen.

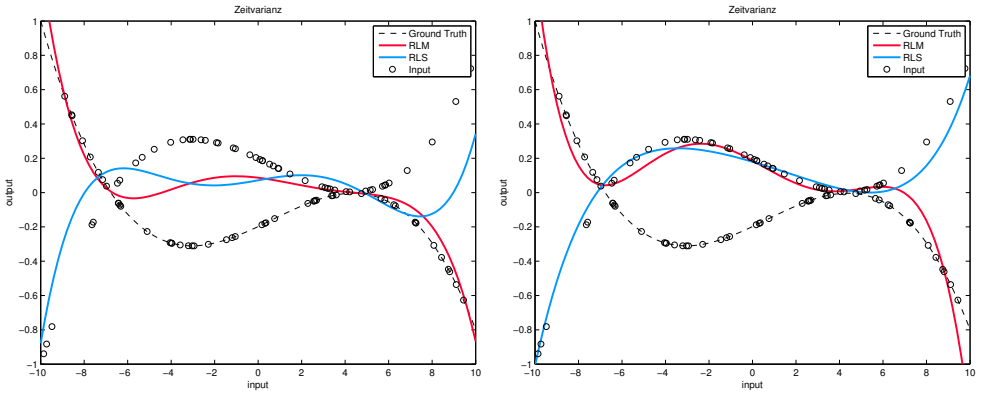
3.4 Zeitvarianz

In diesem Abschnitt soll das Verhalten bei zeitvarianten Zielfunktionen erörtert werden. Dazu wird wird ein Setup[8] mit zwei verschiedenen Vergessenheitsfaktoren λ durchgeführt um dessen Einfluss auf die Approximation zu zeigen. Der Parameter wird in jeder Variante sowohl für den RLM als auch RLS gesetzt. Es ergeben sich die Approximationen in den Abb. (8a) und (8b) mit jeweils $\lambda = 0.99$ (Standard-Setup) und $\lambda = 0.95$.

[6]**Setup**: `Zielfunktion=linear, # Datensätze=100, # Groundtruth=100, noise=0.0, minpath=true, Approximator=Polynom Grad 1`

[7]**Setup**: `Zielfunktion=linear, # Datensätze=100, # Groundtruth=100, noise=0.0, minpath=true, Approximator=Polynom Grad 3`

[8]**Setup**: `Zielfunktion=relearn, # Datensätze=100, # Groundtruth=100, noise=0.0, minpath=false, Approximator=Polynom Grad 5`



(a) $\lambda = 0.99$ (b) $\lambda = 0.95$

Abbildung 8: Approximation und Performance-Ergebnisse bei polynomiellen Approximator 5. Grades auf zeitvarianter Zielfunktion mit unterschiedlichem λ.

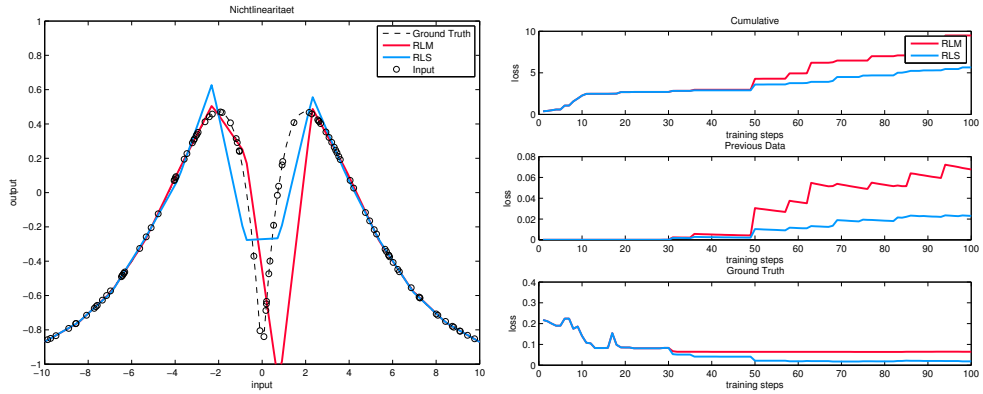
In diesem Szenario kommt die erste Hälfte der Lerndaten ($= 50$) von der gestrichelten Zielfunktion und die zweite Hälfte von ihrer Spiegelung an der x-Achse (schemenhaft an den Punkten in beiden Graphen erkennbar).

Es lässt sich sagen, dass dieses Szenario eine Herausforderung für RLS und RLM darstellt, da diese den Fehler auf allen Lerndaten zu minimieren. Folglich versuchen beim Umschalten der Zielfunktion beide Lernverfahren die alte Zielfunktion abzubilden, werden jedoch durch erhöhte Fehlerraten dazu geführt sich der neuen Zielfunktion anzupassen. Dieses Verhalten kann durch variieren des Vergessenheitsfaktors beschleunigt werden. Dazu wird zunächst Abb. (8a) betrachtet. Im Intervall $[-6, 8]$ verhalten sich beide Graphen relativ ähnlich, wohingegen in den Randgebieten der RLS die neue Zielfunktion und der RLM die alte Zielfunktion abbildet. Erneut liegt dies an der Fehlergröße, welche in den Randgebieten, aufgrund der großen Distanz zwischen den Lerndaten der alten und neuen Zielfunktion, zu einer Hemmung der Anpassung des RLM sorgt. Auch ein kleinerer λ-Wert hat hier keinen Einfluss auf dieses Verhalten, da, wie in Abb. (8b) erkennbar, zwar im mittleren Teil eine klare Näherung der neuen Funktion stattfindet, in den Randgebieten jedoch keine essentielle Änderung zu sehen ist. Genau diese Situation ist problematisch für den RLM. Denn der Teil, der gut vom RLM abgebildet wird ist viel größer als der Teil der schlecht vom RLM abgebildet wird. Folglich werden die Fehlerwerte für die meisten Lerndaten relativ gering sein, was zu einer geringen Standardabweichung führt. Dementsprechend entsteht in den Randgebieten das Problem, dass den dortigen Fehlern immer weniger Bedeutung im Parameter- und Adaptionsupdate zugesprochen wird, da die Wahrscheinlichkeit mit der diese Fehler $\lceil \frac{N}{2} \rceil$-Mal kurz hintereinander auftreten, aufgrund der geringen Größe der problematischen Bereiche, gering ist. Diesem Problem kann entegegengewirkt werden, indem N kleiner gewählt wird (Erklärung zu diesem Sachverhalt analog in Abschnitt (3.1)).

3.5 Nichtlinearität

In diesem Abschnitt geht auf das Verhalten des RLM bei nichtlinearen Zielfunktionen ein. Im Setup[9] wird ein lokaler Approximator mit 14 Stützstellen auf einer nichtlinearen Zielfunktion angewendet. Wie in Abb. (9b) zu sehen ist, sind die Loss-Werte zwar unterschiedlich, spielen sich jedoch in einem relativ niedrigen Zahlenbereich ab.

[9]**Setup**: `Zielfunktion=nonlin, # Datensätze=100, # Groundtruth=100, noise=0.0, minpath=false, Approximator=GLT lin`



(a) (b)

Abbildung 9: Approximation und Performance-Ergebnisse bei lokalem linearem Approximator auf nichtlinearer Zielfunktion.

Das unterschiedliche Verhalten beider Verfahren hängt erneut mit der fehlerhaften Erkennung von Impulsen in den regulären Lerndaten zusammen.

Auf ein weiteres Setup[10] mit einem globalen Approximator 3. Grades wird hier verzeichtet und lediglich auf die Ergebnisse im Anhang verwiesen, da daraus keine neuen Erkentnisse zu ziehen sind (siehe Abb. (15) im Anhang).

3.6 Impulse

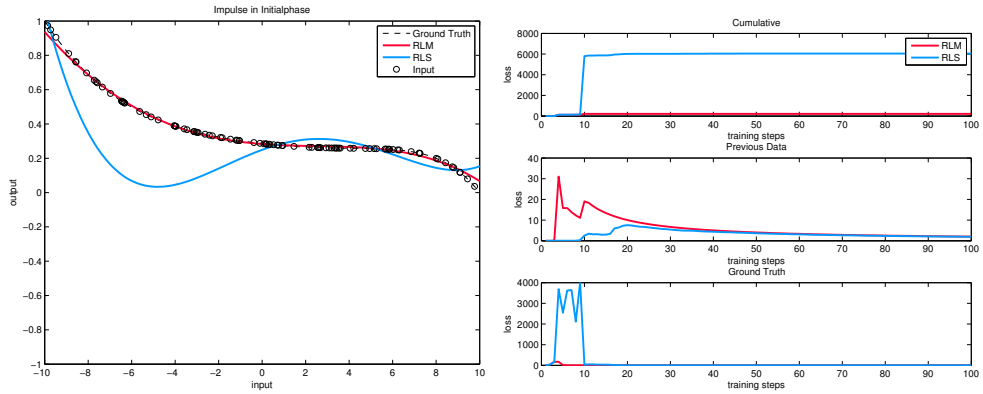
Dieser Teil der Ausarbeitung behandelt das Verhalten des RLM auf impulsversetzten Daten. Wenn vereinzelt Ausreißer in den Lerndaten vorkommen, so werden diese aufgrund des hohen auftretenden Fehlers e_n ignoriert. Der Umgang des RLM mit hohen Fehlern wurde mehrfach in den vorigen Abschnitten diskutiert, daher werden hier zwei Sonderfälle betrachtet, welche Grenzen und Besonderheiten des RLM aufzeigen. Zuerst werden Impulse betrachtet, welche in der initialen Phase des RLM auftreten und anschließend die Wirkung von aufeinanderfolgenden Ausreißern diskutiert.

3.6.1 Impulse in Initialphase

Als Setup[11] wird ein polynomieller Approximator 4. Grades auf eine polynomielle Zielfunktion angewendet. Die Lerndaten zu den Zeitpunkten 4 und 10 sind jeweils mit einem Offset von -10 versehen worden und befinden sich an den Stellen $(-3.2080|-10.3208)$ und $(-7.1535|-10.7153)$. In Abb. (10a) ist sichtbar, dass der RLS die Zielfunktion schlechter annähert als der RLM.

[10] **Setup**: `Zielfunktion=nonlin, # Datensätze=100, # Groundtruth=100, noise=0.0, minpath=false, Approximator=Polynom Grad 3`

[11] **Setup**: `Zielfunktion=poly, # Datensätze=100, # Groundtruth=100, noise=0.0, minpath=false, Approximator=Polynom Grad 4`



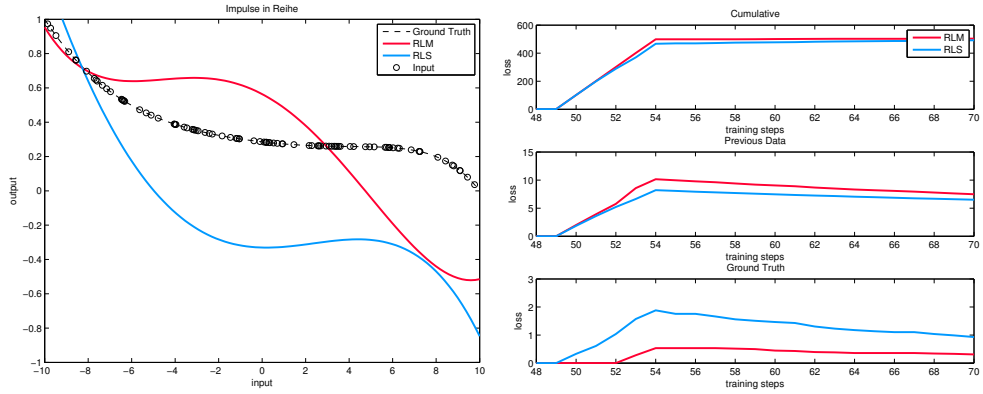
(a) (b)

Abbildung 10: Approximation und Performance-Ergebnisse bei polynomiellem Approximator 4. Grades auf impulsversetzten Daten in der initialen Phase.

Der RLS versucht den quadrierten Fehler zu allen Daten zu minimieren, also auch zu den Impulsen. Der RLM hingegen ignoriert diese beiden Ausreißer. In Abb. (10b) lässt sich das Verhalten beider Algorithmen nachvollziehen. Zu Zeitpunkt 4 und 10 wird der Loss des RLM auf den vergangenen Daten hoch, da er diese nicht abbildet. Der Loss auf die GroundTruth-Daten bleibt dementsprechend gering. Beim RLS verhalten sich die Performancemaße genau andersherum. Durch das Abbilden der Impulse bleibt der Previous-Data Loss gering, jedoch steigt der Loss auf den GroundTruth-Daten stark an. Dadurch entstehen sehr hohe Loss-Werte in der Anfangsphase, welche sich im Laufe der restlichen Lerndaten relativieren. Jedoch ist nach 100 Lerndaten die Approximation des RLS immer noch nicht so gut wie die des RLM.

3.6.2 Impulse in Reihe

Als Setup[12] wird erneut ein polynomieller Approximator 4. Grades auf eine polynomielle Zielfunktion angewendet. Die Lerndaten zu den Zeitpunkten 50, 51, 52, 53 und 54 sind jeweils mit einem Offset von -10 versehen worden und befinden sich an den Stellen $(0.7571|-9.7239)$, $(5.2584|-9.7441)$, $(-1.9125|-9.6784)$, $(-2.9911|-9.6487)$ und $(8.0040|-9.8008)$



(a) (b)

Abbildung 11: Approximation und Performance-Ergebnisse bei polynomiellem Approximator 4. Grades auf impulsversetzten Daten in Reihe.

[12]**Setup**: `Zielfunktion=poly, # Datensätze=100, # Groundtruth=100, noise=0.0, minpath=false, Approximator=Polynom Grad 4`

In diesem Szenario wird auf weitere Grenzen des RLM mit impulsversetzten Daten eingegangen. Dazu wird $N = 6$ gesetzt um dessen Einfluss auf diese Situation zu zeigen. An Abb. (11b) ist zu erkennen, dass mit Beginn der Impulsreihe der RLS einen ansteigenden Previous-Data Loss aufweist. Der RLM hingegen zeigt hier keinen Anstieg. Ferner ist zu beobachten, dass zu Zeitpunkt 53, also beim 4. Lerndatum auch beim RLM ein Anstieg zu verzeichnen ist. Grund hierfür ist das klein gewählte N, welches dazu führt, dass in Reihe vorkommende große Fehler nun vom Median auch beachtet werden, da diese 4 von 6 Elementen in ihm ausmachen. In der Approximation in Abb. (11a) ist daher auch zu erkennen, dass der RLS auf Höhe aller Impulse nach unten gezogen ist, der RLM hingegen nur bei den letzten Impulsen eine negative Abweichung aufweist. Durch die Wahl eines größeren N, kann diesem Effekt entgegengewirkt werden.

3.7 Bewertung

In diesem Abschnitt werden die Inhalte der einzelnen Szenarien zusammengefasst und dazu Stellung genommen. Außerdem wird ein weiteres robuste Lernverfahren kurz vorgestellt und mit dem RLM in Beziehung gesetzt.

Für die Bewertung des RLM gilt die Annahme für potenzielle Szenarien, dass Ausreißer/Impulse in den Lerndaten *nicht* in die Approximation miteinbezogen werden sollen. Sofern diese Annahme nicht gelten soll, ist von der Verwendung des RLM abzuraten.

Insgesamt lässt sich sagen, dass der RLM es schafft einzelne Impulse in Daten zu erkennen und diese nicht in sein Parameter- und Adaptionsupdate miteinzubeziehen. Genau diese Eigenschaft ist in vielen Fällen jedoch problematisch. Denn vorallem in der Anfangsphase werden hohe Fehlerwerte, welche zu Beginn als normal zu betrachten sind, als Ausreißer interpretiert. Dadurch verzögert sich die Annäherung an die Zielfunktion. Im Gegensatz dazu werden Impulse, welche zu Beginn auftreten auch korrekt ignoriert. In den hier vorgestellten Szenarien spielen sich die Unterschiede in der Approximationsgeschwindigkeit zwischen RLM und RLS in einem Rahmen von ca. 10 Lerndaten ab. Sofern auf die Impulserkennung in der initialen Phase nicht verzichtet werden kann, muss das verzögerte Einpendeln in Kauf genommen werden. Wird an Stelle eines globalen, ein lokaler Approximator verwendet, so könnte die Adaption gerade in der Anfangsphase schneller laufen. Dazu muss die Standardabweichung in Abhängigkeit zu der Differenz zwischen der Nulllinie des Approximators und den präsentierten Daten gewählt werden. Wie in den 4 Graphen in Abb. (3) erkennbar ist, verhält sich die Annäherung des RLM im Laufe des Lernprozesses unruhig. Damit ist gemeint, dass die Kurven zu zwei aufeinanderfolgenden Zeitpunkten stark voneinander abweichen können. Bei lokalen Approximatoren wird dieses Verhalten abgeschwächt, da die Funktionsabschnitte beispielsweise nicht ins Unendliche wachsen können.

Generell ist die Performance des RLM am besten gegenüber dem RLS, wenn die Daten Ausreißer enthalten, welche nicht in die Approximation miteinbezogen werden sollen. In den anderen vorgestellten Szenarien hat der RLM eine schlechtere bzw. gleiche Performance wie der RLS aufgezeigt. Bei zeitvarianten Zielfunktionen ist von der Verwendung des RLM eher abzuraten, da dort die Wahrscheinlichkeit einer Fehlinterpretation hoher Fehler groß ist.

3.7.1 Proposed Robust RLS (PRRLS)

Ein anderer Ansatz für ein robustes Lernverfahren wird mit dem *Proposed Robust RLS (PRRLS)* [4] verfolgt. Kern des Verfahrens ist die L_1-Norm des Kreuzkorrelationsvektors einzudämmen. Ein hoher Fehler e_n führt zu einem Anstieg der L_1-Norm von $\|P_n - \lambda P_{n-1}\|$, was dem Algorithmus als Indikator dient Ausreißer zu kennen. Wie bereits erwähnt wird

nun versucht die L_1-Norm einzudämmen indem diese einen gewissen Upper Bound γ_n nicht überschreiten darf.

$$\|P_n - \lambda P_{n-1}\| \leq \gamma_n = \left|\frac{d_n}{e_n}\right|$$

Bewerkstelligt wird diese Bedingung mittels einer Variablen

$$\delta_n = \min\left\{1, \frac{1}{\|x_n e_n\|_1}\right\}$$

welche im Update von S_n und α_n vorkommt

$$S_n = \frac{1}{\lambda}\left(S_{n-1} - \frac{1}{\frac{\lambda}{\delta_n} + x_n^T S_{n-1} x_n} S_{n-1} x_n x_n^T S_{n-1}\right) \tag{10}$$

$$\alpha_n = \alpha_{n-1} + \frac{1}{\frac{\lambda}{\delta_n} + x_n^T S_{n-1} x_n} S_{n-1} x_n e_n \tag{11}$$

Wenn der Fehler klein ist, dann gilt $\frac{1}{\|x_n e_n\|_1} \geq 1$ also $\delta_n = 1$, was dazu führt, dass der PRRLS sich genau wie der RLS verhält. Sollte ein impulsversetztes Datum auftreten, gilt $\frac{1}{\|x_n e_n\|} < 1$ also $\delta_n = \frac{1}{\|x_n e_n\|_1}$ was zu einer Hemmung des Parameterupdates führt (siehe Bruch in Gleichungen (10) und (11)).

Während der RLM die letzten N Fehler als Grundlage für die Impuls-Interpretation verwendet, findet beim PRRLS ein Vergleich seiner aktuellen Approximation mit dem aktuellen Fehler statt. Dies hat zur Folge, dass der RLM bei vielen großen Fehlern seine Grenzen dementsprechend anpasst und diese großen Fehler approximiert. Der PRRLS macht die Toleranz nicht anhand der letzten N Fehler fest. Im Szenario aus Abschnitt (3.6.2) hat der PRRLS einen Vorteil gegenüber dem RLM, da beim PRRLS die Länge der Impulsreihe keine Rolle spielt. Sobald beim RLM hingegen $\lceil \frac{N}{2} \rceil$ Impulse präsent waren, versucht er ab dem nächsten Impuls diese auch abzubilden.

Die Verwendung des PRRLS sollte also in Betracht gezogen werden, wenn nicht absehbar ist wie lange eventuell vorkommende Impulsreihen sind. Ist die Länge bekannt kann durch Parametrisierung eines entsprechendem N dem im RLM vorgebeugt werden.

4 Zusammenfassung

Im Rahmen dieser Ausarbeitung wurde der Recursive Least M-Estimate Algorithmus, ein RLS ähnliches Lernverfahren vorgestellt. Der RLM hat als Ziel hohe Fehlerwerte aus seinem Parameter- und Adaptionsupdate herauszuhalten und stellt somit ein robustes Lernverfahren dar. Durch Messungen in verschiedenen Szenarien wurde seine Performance mit der des RLS verglichen und abschließend bewertet und dessen Grenzen und Besonderheiten aufgeführt.

Literatur

[1] Y. Zou, S. Chan, and T. Ng, "A recursive least m-estimate (rlm) adaptive filter for robust filtering in impulse noise," *Signal Processing Letters, IEEE*, vol. 7, no. 11, pp. 324–326, 2000.

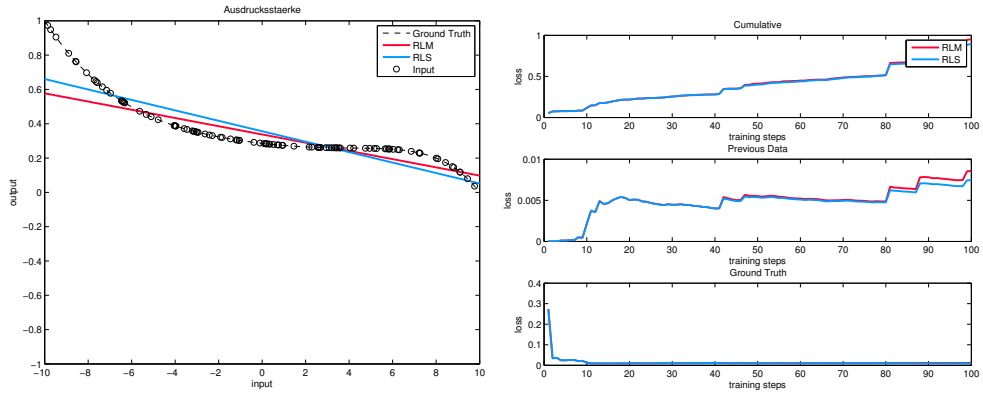
[2] P. S. Diniz, *Adaptive filtering*. Springer, 1997.

[3] A. Buschermoehle, J. Huelsmann, and W. Brockmann, "Uoslib - a library for analysis of online-learning algorithms," in *Proc. 23. Workshop Computational Intelligence*, pp. 355–369, KIT Scientific Publishing, 2013.

[4] M. Z. A. Bhotto and A. Antoniou, "Robust recursive least-squares adaptive-filtering algorithm for impulsive-noise environments," *Signal Processing Letters, IEEE*, vol. 18, no. 3, pp. 185–188, 2011.

A Weitere Graphen/Abbildungen

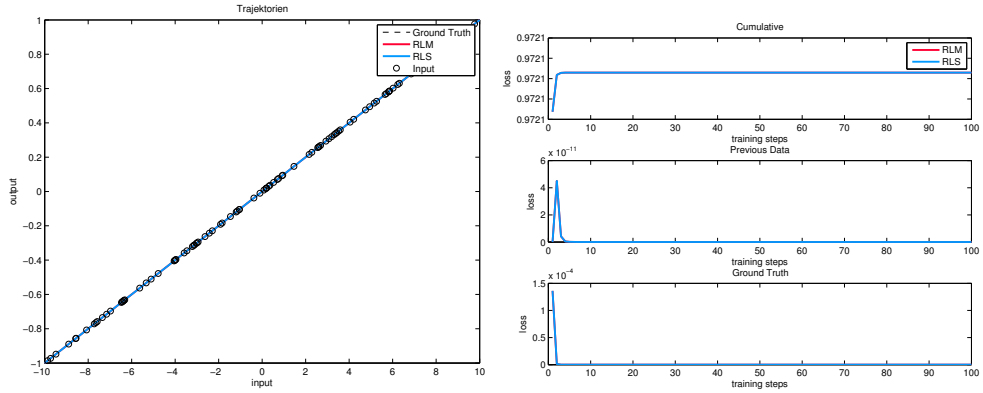
A.1 Graphen zur Ausdrucksstärke



(a) (b)

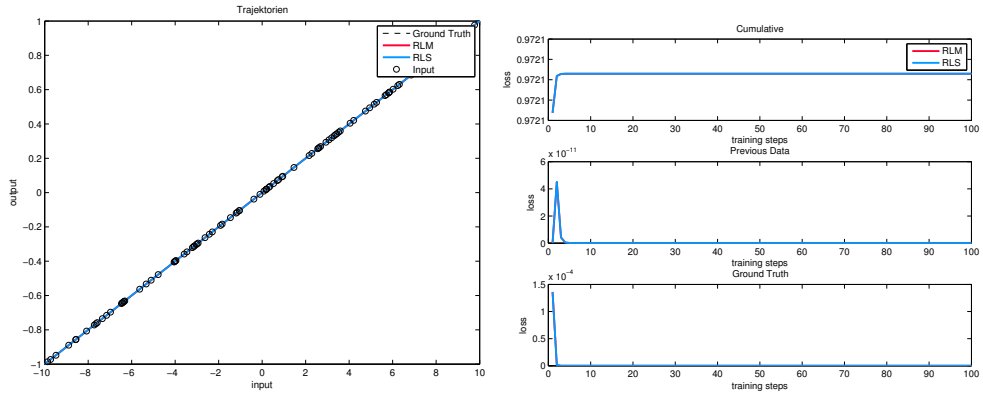
Abbildung 12: Approximation und Performance-Ergebnisse bei linearem Approximator auf polynomiellen GroundTruth-Daten.

A.2 Graphen zu Trajektorieneffekten



(a) (b)

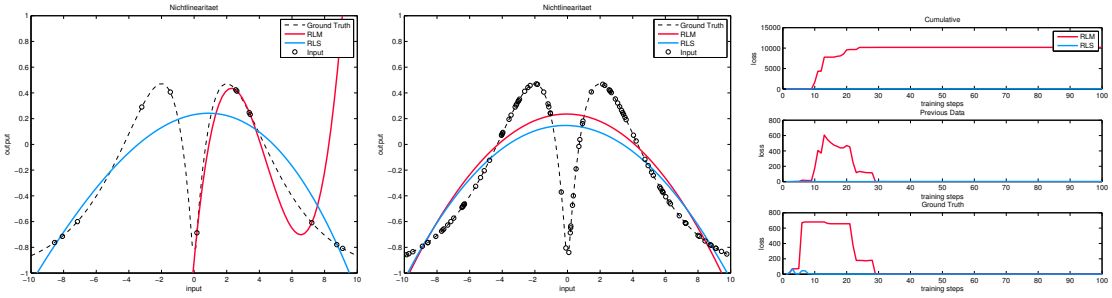
Abbildung 13: Approximation und Performance-Ergebnisse bei linearem Approximator auf linearen GroundTruth-Daten mit Trajektorieneffekten.



(a) (b)

Abbildung 14: Approximation und Performance-Ergebnisse bei polynomiellem Approximator 3. Grades auf linearen GroundTruth-Daten mit Trajektorieneffekten.

A.3 Graphen zu nichtlinearen Zielfunktionen



(a) Approx. nach 13 Daten. (b) Approx. nach 100 Daten. (c) Perf. nach 100 Daten.

Abbildung 15: Approximation und Performance-Ergebnisse bei polynomiellen Approximator 3. Grades auf nichtlinearen GroundTruth-Daten.

B Matlab Code

B.1 Initialisierungsmethode des RLM

icl_initILS_RLM.m

```
% Reference:
% Zou, Y.; Chan, SC.; Ng, TS.: Recursive Least M-estimate (RLM) adaptive
% filter for robust filtering in impulse noise. IEEE Signal Processing
% Letters, 2000, v.7 n. 11, p.324-326
%
% @author Christian Heiden (cheiden@uos.de)
% @date 2014-11-21
%
%
% @input ILS - matlab structure describing the learning system
%
% @input dim - integer > 0, number of dimensions
%
% @input algSetup - algorithm specific setup parameters
%
% @return ILS - matlab structure describing the learning system

function ILS = icl_initILS_RLM(ILS, dim, algSetup)

    % if either algSetup is not provided or empty (check by looking for
```

```
21      % algSetup.S)
22      if(nargin < 3 || isfield(algSetup, 'S') == 0) %default setup
23
24          %covariance matrix
25          ILS.S = 10^5*eye(length(ILS.phi));
26          %forgetting factor
27          ILS.g = 1;
28
29          %-- M-estimate function --%
30          % no. of history entries for calc. of new boundaries
31          ILS.errhist = 30;
32          % start value of deviation for calculation of new boundaries
33          ILS.dev = 1;
34          % forgetting factor that is used in calc of new boundaries
35          ILS.gdev = 0.9;
36
37      else
38
39          %covariance matrix
40          ILS.S = algSetup.S*eye(length(ILS.phi));
41          %forgetting factor
42          ILS.g = algSetup.g;
43
44          %-- M-estimate function --%
45          % see comments above for description
46          ILS.errhist = algSetup.errhist;
47          ILS.dev = algSetup.dev;
48          ILS.gdev = algSetup.gdev;
49      end
50
51      % initilize array for errorhistory (this will work like a queue later
52      % on)
53      ILS.errs = zeros(1,ILS.errhist);
54
55      % set bounds dependant on deviation
56      ILS.eps = 1.96 * ILS.dev;
57      ILS.delta_1 = 2.24 * ILS.dev;
58      ILS.delta_2 = 2.576 * ILS.dev;
59  end
```

B.2 Lernmethode des RLM

icl_learn_RLM.m

```
1  % Reference:
2  % Zou, Y.; Chan, SC.; Ng, TS.: Recursive Least M-estimate (RLM) adaptive
3  % filter for robust filtering in impulse noise. IEEE Signal Processing
4  % Letters, 2000, v.7 n. 11, p.324-326
5  %
6  % @author Christian Heiden (cheiden@uos.de)
7  % @date 2014-11-21
8  %
9  % @input ILS - structure describing the learning system
10 %
11 % @input x - double input vector in [-10,10]
12 %
13 % @input y - true label, i.e. target value of the example, in [-1, 1] for
14 %            regression or [-1; 1] for classification
15 %
16 % @input yp - predicted label, i.e. current output of the approximation, in
17 %             [-1, 1] for regression or [-1; 1] for classification
18 %
19 % @input mode - operating mode (regression=1, classification=2)
20 %
21 % @return ILS - updated matlab structure describing the learning system
22
23 function ILS = icl_learn_RLM(ILS, x, y, yp, mode)
24
25     % calculation of error between prediction and real value. It is often used
26     % and therefore calulculated here for better readability
27     e_n = y-yp;
28
29     % convert x to parameter space
30     phix = icl_transform(x, ILS);
31
```

```
32      %-- update parameter vector --%
33      % get value of derivation of M-estimate function (compare to q(e(n)) in
34      % reference)
35      q_n = m_estimate(e_n, ILS);
36
37      % Calculate gain vector
38      gain = (q_n*ILS.S*phix')/(ILS.g+q_n*phix*ILS.S*phix');
39
40      % newAlpha = ILS.alpha + error * k_n;
41      newAlpha = ILS.alpha + (gain * e_n)';
42
43      %update covariance matrix
44      ILS.S = 1/ILS.g * (eye(length(phix)) - gain * phix) * ILS.S;
45
46      %update ILS with new parameter vector
47      ILS = icl_updateILS(ILS, newAlpha, phix);
48
49      % Update epsilon, delta_1 an delta_2
50      ILS = update_boundaries(e_n, ILS);
51
52      if(mode == 2)
53          warning(['RLM is designed for regression. For classification AROW'...
54              'is the more appropriate variant.']);
55      end
56
57  end
58
59  %-- Derivation of M-Estimate function --%
60  %
61  % This function is used to downweight impulse noise on the learning step.
62  % The value corresponds to the function q(e) = derivative(rho)/e on page
63  % 325 (left). Rho is the M-estimate function (page 324 right).
64  %
65  % @error    - error of predicted value and real value
66  %
67  % @ILS      - data structure used to describe the learning system
68  %
69  % @return   - The higher error is, the smaller the output y will be. The
70  % result of this function is used to downweight the changes noisy data will
71  % have on our learning step. In other words, the higher the error, the less
72  % our algorithm will adapt to this error.
73  % The boundaries eps, delta_1 and delta_2 are important parameters for this
74  % function since they manipulate the 'amount' of adaption for certain
75  % errorvalues. These parameters als also adapted (see function(...)).
76  function y = m_estimate(e_n, ILS)
77
78      % improves readability
79      eps = ILS.eps;
80      delta_1 = ILS.delta_1;
81      delta_2 = ILS.delta_2;
82
83      % absolute value of error (is frequently used)
84      e = abs(e_n);
85
86      % derivatives of partial functions.
87      if (0 < e && e < eps)
88          y = e_n;
89      elseif (eps <= e && e < delta_1)
90          y = eps;
91      elseif (delta_1 <= e && e < delta_2)
92          y = e_n*eps*(e - delta_2)/((delta_1 - delta_2)*e);
93      elseif (delta_2 <= e)
94          y = 0;
95      end
96
97      % normalize y on error (take care of the case when error is zero)
98      if (e_n ~= 0)
99          y = y/e_n;
100     else
101         y = 1;
102     end
103
104 end
105
106 %-- Calculation of new Parameters for M-estimate function --%
107 %
```

```
% This function is used to update the parameters (bounderies) of the
% M-estimate function. The calculation corresponds to chapter IV of the
% reference (p.325 right).
% The calculation of the new parameters depends mainly on the median of the
% old error values that are stored in ILS.errs. ILS.errs is a queue-like
% array that stores the old error values up to a certain time point (now -
% ILS.errhist). In sigma this beforementioned median and also a forgetting
% value ILS.gdev are taken into consideration with a 'constant' C1.
% The new parameters are calculated, by the 3 formulas at the bottom of the
% function.
%
% @ILS      - data structure used to describe the learning system

function ILS = update_boundaries(e_n, ILS)
    % Store error in errorhistory
    % first move all entries to next place and enter new error (like a queue)
    for i = length(ILS.errs)-1:-1:1
        ILS.errs(i+1) = ILS.errs(i);
    end

    % insert new error at the front
    ILS.errs(1) = e_n^2;

    % calc sigma (compare to ref chapter IV)
    C_1 = 1.483*(1+5/(ILS.errhist-1));
    ILS.dev = sqrt( ILS.gdev * ILS.dev^2 + C_1*(1-ILS.gdev) * median(ILS.errs));

    % Calculation of new parameters
    ILS.eps = 1.96 * ILS.dev;
    ILS.delta_1 = 2.24 * ILS.dev;
    ILS.delta_2 = 2.576 * ILS.dev;
end
```

B.3 Impulsgenerierung

icl_genImpulses.m

```
% @author Christian Heiden (cheiden@uos.de)
% @date 2014-11-21
%
%
% @input old_data - the points that are generated so far
%
% @input option - an option to set the different modes of this function
% (still incomplete)
%                   'impulseArray' - varargin holds an array with (pos,
%                   offset) datapairs which is applied to old_data
%
%                   'randomX' - not implemented
%
%                   'randomY' - not implemented
%
%                   'random' - not implemented
%
% @input varargin - set with additional input dependant on the option
%
% @return impulse_data which basically is old_data with Impulses

function impulse_data = icl_genImpulses(old_data, option, varargin)

    imp_vals = [];

    if (option == 'impulseArray')
        imp_vals = varargin{1}
    elseif (option == 'randomX')
        % INCOMPLETE
        imp_vals(:,1) = round(rand(length(varargin{1}),1)*length(varargin{1}))
        imp_vals(:,2) = varargin{1}
    elseif (option == 'randomY')
        % INCOMPLETE
        imp_vals(:,1) = varargin{1}
    elseif (option == 'random')
        % INCOMPLETE
        percX = varargin{1}
        percY = varargin{2}
```

```
39          imp_no = length(bla) * percX
40          imp_vals(:,1) = round(rand(imp_no,1)*length(varargin{1}))
41          imp_vals(:,2) = round(rand(imp_no,1) * percY)
42      end
43
44      impulse_data = old_data;
45      impulse_data(imp_vals(:,1),2) = old_data(imp_vals(:,1),2) + imp_vals(:,2)
46
47  end
```