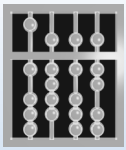


Institut für
Informatik

TECHNISCHE
UNIVERSITÄT
MÜNCHEN

Neuronale Netze zur Modellbildung in der Regelungstechnik

Michael Sturm

Neuronale Netze zur Modellbildung in der Regelungstechnik

Michael Sturm

Institut für Informatik
der Technischen Universität München

Neuronale Netze zur Modellbildung in der Regelungstechnik

Michael Sturm

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr. Arndt Bode
Prüfer der Dissertation:	
1.	Univ.-Prof. Dr. Dr.h.c. Wilfried Brauer
2.	Univ.-Prof. Dr. Raùl Rojas, Freie Universität Berlin

Die Dissertation wurde am 23.9.1999 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 25.2.2000 angenommen.

Danksagung

Mein ganz besonderer Dank gilt meinem Doktorvater Herrn Professor Dr. Dr.h.c. Wilfried Brauer, für seine stete Förderung und Unterstützung dieser Arbeit. Durch seine kritischen Fragen und Anmerkungen verhalf er mir zu immer neuen Einblicken in das zu bearbeitende Thema. Stets ließ er mir freie Hand bei der Realisierung von Ideen, wobei mir seine kontinuierliche Unterstützung sicher war.

Herrn Professor Dr. Raùl Rojas möchte ich für die Übernahme des Koreferats meinen Dank aussprechen.

Die Kollegen der Forschungsgruppe KI/Kognition stellten den akademischen, technischen und oft auch privaten Rahmen dieser Arbeit. Insbesondere Herrn Dr. Till Brychcy möchte ich für seine kontinuierliche Unterstützung und vielen anregenden Diskussionen danken. Durch die mühevolle Arbeit des Korrekturlesens hat Frau Claudia Ungerer zu einer annehmbaren Form dieser Arbeit beigetragen. Herrn Clemens Kirchmair bin ich für die umfangreichen Hysterese-Daten dankbar, die mir zum Testen des LEMON-Verfahrens dienten. Herrn Dr. Gerhard Weiß danke ich für den Einblick in den aktuellen Stand der Multiagentensysteme.

Mein Dank gebührt auch unseren Projektpartnern im Projekt ACON[1], die dazu beigetragen haben, mein Verständnis der Modellbildungsproblematik aus der Sicht der Regelungstechnik zu vertiefen. Ich möchte besonders Herrn Dr. Klaus Eder und Herrn Dirk Stübener von Kratzer Automation AG für zahllose Diskussionen und anregende Gespräche danken. Das Überlassen realer Meßdaten des Instituts für Elektrische Energietechnik - Antriebstechnik der Universität Gesamthochschule Kassel durch Herrn Dr.-Ing. M. Ayeb gab mir die Gelegenheit die On-Line-Fähigkeit von LEMON auszutesten und zu verfeinern.

Den letzten Schliff erhielt meine Arbeit durch die intensive Korrektur und zahlreichen Anregungen von Frau Leni Kinder, wofür ich mich herzlich bedanke.

Zuletzt möchte ich ganz herzlich meiner Frau, Dr. Margit Sturm, für Ihre Unterstützung, sowohl in fachlichen als auch persönlichen Belangen, danken.

[1]Diese Arbeit wurde vom Bundesministerium für Bildung, Wissenschaft, Forschung und Technologie im Rahmen des Projektes ACON (FKZ 01IN510C8) gefördert.

Kurzfassung

Modellbildung in der Regelungstechnik bedeutet die möglichst gute Approximation eines technischen Prozesses durch ein mathematisches bzw. physikalisches Modell. Für Prozesse, die einem linearen Differentialgleichungssystem gehorchen, existiert hierfür eine einheitliche Theorie. Ist diese Linearitätsbedingung nicht erfüllt, ist das System also nichtlinear, so stellt die Modellierung eine große Herausforderung dar. Speziell die On-Line-Fähigkeit der Modellierung ist meist nicht erfüllbar.

In dieser Arbeit wird nach einer kurzen Einführung in die Grundlagen der Regelungstechnik eine neue, vereinheitlichte Sichtweise gängiger Modellbildungsverfahren gegeben, die insbesondere das Feld der neuronalen Netze umfaßt. Schwächen der verwendeten Verfahren werden aufgezeigt und mögliche Lösungsansätze diskutiert.

Aufbauend auf der Forderung nach „On-Line-Fähigkeit“ und „Modellierung nichtlinearer Systeme“ wird der Algorithmus LEMON (**L**ocal **E**llipsoidal **Mo**del **N**etwork) entworfen. Dieser basiert einerseits auf der Verwendung einer ellipsoiden Kartierungsmethode mit einer neu entwickelten ellipsoiden Metrik zur Repräsentation des Prozeßzustandsraumes. Andererseits erlaubt die ellipsoide Karte die Zuordnung lokaler Modelle zu einzelnen Zustandsraumbereichen, wobei fast beliebige Modelltypen genutzt werden können. Dank eines hierfür entworfenen intelligenten Modellselektionsverfahrens ist eine lokale Anpassung der Modellkomplexität an die Prozeßkomplexität möglich.

Mit der Verfügbarkeit von LEMON stellt sich die Frage nach geeigneten Filteralgorithmen, um auch verrauschte Daten mit Ausreißern und Sprüngen im Nutzanteil des Eingabesignals behandeln zu können. Hierzu wird eine neue Art von Regressionsfilter entworfen, der mit Hilfe einer ebenfalls neu entwickelten Sprungerkennung einzelne Ausreißer von tatsächlichen Sprüngen der Originaldynamik trennt.

Ein Vergleich des entwickelten Filterverfahrens mit klassichen Filtern und die Präsentation verschiedener Simulationsergebnisse von LEMON schließen diese Arbeit ab.

Inhaltsverzeichnis

Kurzfassung **iii**

1 Einleitung **1**
1.1 Zielsetzung . . . 1
1.2 Aufbau der Arbeit . . . 2

2 Regelungstechnik **5**
2.1 Einführung . . . 5
2.1.1 Begriffsdefinitionen . . . 5
2.2 Probleme bei der Regelung . . . 10
2.3 Modelle in der Regelung . . . 11
2.3.1 Einsatz von Gesamtprozeßmodellen . . . 11
2.3.2 Neutralisierung einzelner Nichtlinearitäten . . . 13

3 Modellbildung **15**
3.1 Der Modellbegriff . . . 15
3.2 Lernverfahren . . . 16
3.2.1 Das Bias-Variance-Problem . . . 17
3.2.2 Belernen Allgemeiner Modelle . . . 20
3.2.3 Belernen differenzierbarer Modelle . . . 20
3.3 Neuronale Netze . . . 23
3.3.1 Einfache vorwärtsgerichtete Netze . . . 23
3.3.2 Rekurrente Netze . . . 27
3.3.3 Andere Netzarchitekturen . . . 30
3.3.4 Ansätze zur Vermeidung des Bias-Variance-Problems . . . 33
3.4 Probleme bei der Modellbildung . . . 34
3.4.1 Behandlung von Rauschen, Ausreißern und Sprüngen . . . 34
3.4.2 Extrapolation bei unbekannten Eingaben . . . 36
3.4.3 Unvorhergesehenes Vergessen bei kontinuierlichem Lernen . . . 36

4 Datenerfassung **37**
4.1 Filterung . . . 37
4.1.1 Einführung verschiedener Filtertypen . . . 37
4.1.2 Beispiele klassischer IIR-Filter . . . 38
4.1.3 Modellbasierte FIR-Filter . . . 39
4.1.4 Probleme durch Ausreißer und Sprünge . . . 43
4.1.5 Sprungerkennung anhand von Varianzkriterien . . . 44
4.1.6 Regressionsfilter mit Sprungerkennung . . . 47
4.2 Auffinden optimaler Meßpunkte . . . 50

4.2.1 Kostenfunktion . . . 50
4.2.2 Ansätze . . . 50
4.2.3 Probleme . . . 51

5 LEMON - Ein On-Line-Verfahren zur lokalen Modellbildung 53
5.1 Ellipsoide Karten . . . 54
5.1.1 Definition eines Ellipsoids . . . 54
5.1.2 Definition der Zugehörigkeit zu einem Ellipsoid . . . 56
5.1.3 Das heuristische Abstandsmaß . . . 56
5.1.4 Das modifizierte heuristische Abstandsmaß . . . 57
5.1.5 Das korrekte Abstandsmaß . . . 58
5.1.6 Komplexitätsreduktion des korrekten Abstandsmaßes . . . 61
5.2 On-Line-Clustering mit Ellipsoiden Karten . . . 65
5.2.1 Räumliche Adaption von Ellipsoiden . . . 68
5.3 Die lokale Modellbildungskomponente . . . 69
5.3.1 Das Konfidenzmaß . . . 72
5.3.2 Die Modellauswahlkomponente . . . 73

6 Simulationsergebnisse 77
6.1 Regressionsfilter mit Sprungerkennung . . . 77
6.1.1 Versuchsaufbau . . . 77
6.1.2 Test der Sprungerkennung . . . 78
6.1.3 Vergleich mit klassischen Filterverfahren . . . 78
6.1.4 Beurteilung . . . 80
6.2 Modellbildung mit LEMON . . . 81
6.2.1 Künstliche Beispielfunktion . . . 81
6.2.2 Hysterese . . . 84
6.2.3 Prüfstandssimulation . . . 89
6.2.4 Vergleich verschiedener Verfahren am Heizungsmodell . . . 91

7 Schlußbetrachtung 95
7.1 Zusammenfassung . . . 95
7.2 Abgrenzung . . . 96
7.2.1 Multiagentensysteme . . . 96
7.2.2 „Mixture of Experts“-Architekturen . . . 97
7.2.3 Intelligente hybride Systeme . . . 98
7.3 Ausblick . . . 98

A Simulation einer Gebäudeheizung 101
A.1 Technische Beschreibung . . . 101
A.1.1 Brenner . . . 102
A.1.2 Heizkessel . . . 102
A.1.3 Pumpe . . . 103
A.1.4 Leitung . . . 104
A.1.5 Heizkörper . . . 104
A.1.6 Raumthermostat . . . 106
A.1.7 Wohnraum . . . 106
A.2 Simulationsbeispiele . . . 109
A.2.1 Energiefluß durch eine Wand bei konstanter Innen- und Außentemperatur 109

A.2.2 Ein kompletter Zwei-Tages Heiz-Zyklus 111

B LEMON-Algorithmus **115**

C Parameterbeschreibung **117**
C.1 Simulationssteuerung von LEMON 117
C.2 Fuzzy-Variablen 118
C.2.1 Modellspezifische Variablen 118
C.2.2 Datenspezifische Variablen 118

Literatur **119**

Index **127**

Abbildungsverzeichnis

2.1 Blockdarstellung eines Übertragungsgliedes 6
2.2 Typische nichtlineare Übertragungsfunktionen 7
2.3 Standardblockschaltbild des Regelkreises 9
2.4 Sprung- und Anstiegsantwort eines PID-Reglers 10
2.5 Generierung eines Prozeßmodells 11
2.6 Nutzung von Prozeßmodellen zur Vorsteuerung 12
2.7 Regelung mit Hilfe von Modellvorhersagen 13
2.8 Egalisierung einer Nichtlinearität 14
2.9 Auslöschung einer Nichtlinearität 14

3.1 Beispiel für das Bias-Variance-Problem 18
3.2 Biologisches Neuron mit Synapse 23
3.3 Verlauf eines Aktionspotentials 24
3.4 Aufbau eines Multi-Layer-Perzeptrons 25
3.5 Aufbau eines „Radial-Basis-Function“-Netzes 26
3.6 Beispiel eines rekurrenten Netzes 27
3.7 Aufbau eines Jordan-Netzes 28
3.8 Backpropagation Through Time 29
3.9 Dynamisches ARMA-Neuron 31
3.10 Adaptive Time Delay Neural Network 31
3.11 Long Short-Term Memory Speicherzelle 33
3.12 Approximation einer Sprungfunktion 35

4.1 Frequenzgang einiger klassischer IIR-Filter 39
4.2 Frequenzgang verschiedener Regressionsfilter (FIR) 44
4.3 Sprungantwort einiger klassischer IIR-Filter 45
4.4 Dichte der Normalverteilung $\phi_{\mu,\sigma}(x)$ 46
4.5 Sprungantwort verschiedener Regressionsfilter (FIR) 48
4.6 Performanz des Regressionsfilters mit Sprungerkennung 49

5.1 Beispiel einer Voronoi-Parkettierung 53
5.2 Überführen eines Kreises in ein Ellipsoid 55
5.3 Näherung des Lotabstandes 57
5.4 Verlauf der Voronoi-Grenze für geschachtelte Ellipsoide 58
5.5 Plot der Voronoi-Grenze zweier Kreise in Abhängigkeit vom Radius 59
5.6 Voronoi-Parkettierung abhängig von der Zahl der Optimierungsschritte 61
5.7 Vergleich der Voronoi-Parkettierung bei Verwendung verschiedener Metriken . 62
5.8 Statistik über Dauer und Zahl der Optimiererschritte zur Distanzberechnung . 63
5.9 Maximaler Fehler $\Gamma_{\max}$ und Winkel $\alpha_{\max}$ für $\eta_2 \equiv 1$. 64

5.10 Nummerische Bestimmung des maximalen Fehlers $\Gamma_{\max}$ 64
5.11 Verwendung ellipsoider Karten zur Zustandserkennung mit Modellvorhersage . 66
5.12 Adaption eines Ellipsoids 68
5.13 Beispiel eines On-Line-Cluster-Laufes (Hysterese) 70
5.14 Benutzergesteuerte Modellwahl bei ellipsoiden Karten mit lokalen Modellen . . 74
5.15 Beispiel einer Fuzzy-Regelbasis, die einfache Modelle bevorzugt 75

6.1 Vergleich eines Chebyshev- und eines Regressionsfilters 78
6.2 Vergleich von Regressionsfiltern mit und ohne Sprungerkennung 79
6.3 Vergleich von Chebyshev- und Regressionsfilter mit Sprungerkennung (1) . . . 80
6.4 Vergleich von Chebyshev- und Regressionsfilter mit Sprungerkennung (2) . . . 80
6.5 Trajektorie und Funktionsfläche der künstlichen Testfunktion 82
6.6 Fehlerfläche der modellierten Testfunktion 83
6.7 Performanz der LEMON-Hysteresemodelle an der Trainingstrajektorie 85
6.8 Performanz der LEMON-Hysteresemodelle an einer Testtrajektorie (1) 86
6.9 Performanz des LEMON-Hysteresemodelle an einer Testtrajektorie (2) 86
6.10 Zustandsraumzerlegung einer Hysterese durch LEMON 88
6.11 MATLAB-Prüfstandsimulation mit PINN-Regler 89
6.12 Ergebnis einer Prüfstandsimulation mit PINN-Regler und LEMON 90
6.13 Performanz von LEMON am Heizungsmodell 92
6.14 Performanz von ATNN- und DRBF-Netzen am Heizungsmodell 92

A.1 Überblick über die Gebäudeheizungssimulation 101
A.2 Schichtenmodell einer Gebäudewand 108
A.3 Energiefluß durch eine Gebäudeaußenwand (Temperatursprung) 110
A.4 Temperaturverteilung in einer Gebäudeaußenwand (Temperatursprung) 110
A.5 Temperatur verschiedener Heizungskomponenten (Zwei-Tages Zyklus) 111
A.6 Energiefluß durch eine Gebäudeaußenwand (Zwei-Tages Zyklus) 112
A.7 Temperaturverteilung in einer Gebäudeaußenwand (Zwei-Tages Zyklus) 112

Tabellenverzeichnis

5.1 Effizienz der Fehlerabschätzung an einem Beispieldatensatz 65

6.1 Von LEMON gewählte lokale Modelle (Künstliche Beispielfunktion) 82
6.2 Von LEMON gewählte lokale Modelle (Hysterese) 85
6.3 Absolute Vorhersage-Fehler der Hysteresemodelle von LEMON 87
6.4 Verbesserung der Reglerperformanz durch Einsatz verschiedener Modelle . . . 90

Kapitel 1

Einleitung

Modellbildung als die Nachbildung eines real existierenden Systems wird mit fortschreitender Technisierung zu einem immer wichtigeren Thema. Lag zu Beginn der Schwerpunkt auf dem Verstehen und Modellieren unserer Umwelt im Sinne der physikalischen Theorie, so rückte mit zunehmender Verfügbarkeit kleiner, leistungsfähiger Rechner ein zweiter Aspekt ins Blickfeld: Die Unterstützung von Regelungen durch Prozeßmodelle. So kann mit Hilfe eines Modells eine Vorhersage der realen Reaktion eines Prozesses auf bestimmte Aktionen gemacht werden, anhand derer die „beste“ Aktion gewählt werden kann.

Für lineare Prozesse, die einem linearen Differentialgleichungssystem gehorchen, ist eine einheitliche Theorie vorhanden, die zum Beispiel auch nicht meßbare Parameter approximieren kann. Sobald diese Linearitätsbedingung aber verletzt wird, der Prozeß also nichtlinear ist, stellt die Modellierung eine große Herausforderung dar. Oft sind die Prozesse entweder so komplex, daß eine mathematische Form der Beschreibung (zum Beispiel in Form eines Differentialgleichungssystems) nicht möglich ist, oder notwendige Modell-Parameter nicht meßbar sind.

Erweitert man die Anforderungen an das Modellbildungsverfahren, so daß zeitvariante Parameter auch während des laufenden Betriebs – also On-Line – nachgelernt werden müssen, so stößt man an die Grenzen derzeit verfügbarer Methoden.

1.1 Zielsetzung

In dieser Arbeit soll aufbauend auf einer neuen, vereinheitlichten Sichtweise gängiger Modellbildungsverfahren ein neues Modellbildungsverfahren mit folgenden Eigenschaften entwickelt werden:

On-Line-Fähigkeit. Das Verfahren soll in der Lage sein, im laufenden Betrieb neues Wissen zu erlernen. Dabei soll bereits vorhandene Information erhalten bleiben. Dies stellt für viele Modellbildungsverfahren ein Problem dar, da diese bereits erlerntes Wissen in nicht vorhersehbarer Weise überschreiben.

Einbringen von Vorwissen. Steht Expertenwissen über das zu modellierende System zur Verfügung, so soll dieses in einfacher Weise integrierbar sein. Ansonsten soll das Verfahren weitgehend autonom ablaufen.

Modellierung nichtlinearer Systeme. Die Approximation verschiedener, nichtlinearer Dynamiken muß möglich sein. Auch Sprungübergänge sollen modellierbar sein.

Generalisierung. Das Verfahren soll nicht bloß „auswendig lernen“, sondern aus gelerntem Wissen abstrahieren können. Dies bedeutet unter anderem, daß die Modellkomplexität an die Prozeßkomplexität bzw. Komplexität der verschiedenen Dynamiken angepaßt werden muß.

Interpretierbar. Aus dem Modell sollen, wenn möglich, neue Erkenntnisse über den zu modellierenden Prozeß ablesbar sein. Dies ist zum Beispiel für Fragen der Stabilität oder zum Entwurf von Reglern wichtig.

Mit der Verfügbarkeit dieses Verfahrens stellt sich die Frage nach geeigneten Filteralgorithmen, um auch verrauschte Daten mit Ausreißern und Sprüngen im Nutzanteil des Eingabesignals On-Line behandeln zu können. Es erscheint also sinnvoll einen Filter zu entwickeln, der einzelne Ausreißer von tatsächlichen Sprüngen und Dynamikwechseln trennt.

1.2 Aufbau der Arbeit

Im folgenden wird ein kurzer Überblick über die weiteren Kapitel dieser Arbeit gegeben. Dabei werden jeweils die eigenen Forschungsbeiträge zu den einzelnen Themenkreisen genannt.

Kapitel 2. Hier werden zunächst einige Grundlagen der Regelungstechnik eingeführt, um den Begriff der Regelung näher zu bestimmen. Anschließend wird anhand von Problembeispielen demonstriert, wie Modelle zur Verbesserung bestehender Regelalgorithmen eingesetzt werden können. Dies motiviert die Entwicklung von Modellbildungsverfahren.

Kapitel 3. Basierend auf einem mathematisch/technischen Modellbegriff wird eine neue, einheitliche Sichtweise der Modellbildung präsentiert. Dazu werden zuerst verschiedene allgemeine Lernverfahren diskutiert, um dann das Feld der neuronalen Netze innerhalb dieses Rahmens zu präsentieren. Das Kapitel schließt mit der Erörterung von Problemen beim Einsatz der vorgestellten Modellbildungsverfahren.

Kapitel 4. Um das Modell eines technischen Prozesses zu erstellen, muß dieser vermessen werden. Dieses Kapitel beschäftigt sich einerseits mit der dabei notwendigen Vorverarbeitung der erhaltenen Daten, andererseits wird untersucht, inwiefern dabei die Berechnung optimaler Meßpunkte möglich ist. Um die Vorverarbeitung On-Line durchführen zu können, wird ein neues effizientes Filterverfahren entwickelt, das selbstständig Dynamikwechsel erkennen und diesen folgen kann. Eine neue, *explizite* Form der Pseudoinversen für äquidistante Regressionsprobleme garantiert die Einsatzfähigkeit auch in Rechnern geringer Kapazität.

Kapitel 5. In diesem Kapitel wird ein neues On-Line-Verfahren zur lokalen Modellbildung mit dem Namen LEMON (**L**ocal **E**llipsoidal **Mo**del **N**etwork) entwickelt. Dieses Verfahren zeichnet sich durch die Fähigkeit aus, physikalische Prozesse On-Line zu modellieren. Dabei wird durch eine eigens entwickelte ellipsoide Metrik der Prozeßzustandsraum kartiert. Die einzelnen Bereiche werden dann anhand einer Regelbasis, die prozeßspezifisch vorgegeben werden kann, auf verschiedene lokale Modelle, auch *unterschiedlicher* Architektur, aufgeteilt. Neue Informationen können dadurch im On-Line-Sinne ohne Verlust bisher gelernter Information nachtrainiert werden.

Kapitel 6. Dieser Abschnitt demonstriert die Praxistauglichkeit des entwickelten Filter- und Modellbildungsverfahrens. Einerseits werden dabei mit Hilfe künstlicher Daten die dem Entwurf zugrunde gelegten Forderungen, wie etwa On-Line-Fähigkeit, überprüft. Andererseits finden sich Vergleiche mit anderen Verfahren, zum Beispiel anhand der Performanz bei der Regelung einer komplexen Prüfstandsimulation.

Kapitel 7. In dieser Schlußbetrachtung wird zuerst eine Zusammenfassung der erreichten Ziele gegeben. Anschließend findet sich eine Abgrenzung von LEMON zu ähnlichen Ansätzen. Ein Ausblick auf mögliche weitere Forschungsziele, aufbauend auf den präsentierten Ergebnissen, schließt die Arbeit ab.

Anhang. In Anhang A wird die Simulation einer Gebäudeheizung in MATLAB/SIMULINK beschrieben. Diese wurde insbesondere zum Test von Verfahren zur Totzeiterkennung entwickelt. Sie zeichnet sich durch einen modularen Aufbau aus, so daß verschiedene Testszenarien einfach erstellt werden können. Anhang B enthält den LEMON-Algorithmus als Pseudocode und Anhang C faßt seine Simulationsparameter zusammen.

Kapitel 2

Regelungstechnik

Der Begriff der Regelung ist im täglichen Gebrauch wohl jedem durch die zunehmende Technisierung unserer Gesellschaft vertraut. So gibt es eine Temperaturregelung im Backofen, die Geschwindigkeitsregelung des Autos („Tempomat“) und viele Beispiele mehr. Dennoch macht man sich nicht unbedingt bewußt, was Regelung z.B. von der Steuerung abhebt, bzw. welcher Aufwand nötig ist, um Regelung überhaupt erst zu ermöglichen. Im folgenden sollen einige Grundbegriffe der Regelungstechnik eingeführt werden, um dann anhand von Problembeispielen die vorliegende Arbeit zu motivieren.

2.1 Einführung

Kennzeichnend für eine *Regelung* ist ein geschlossener Wirkungskreis. Dabei soll eine physikalische Größe, die sogenannte *Regelgröße* einer technischen Anlage einem bestimmten zeitlichen Verlauf folgen, was auch heißen kann, daß diese Größe konstant gehalten werden muß. Dazu stehen in allgemeinen Eingriffsmöglichkeiten aufgrund von verstellbaren Parametern, den *Stellgrößen* , zur Verfügung. Die Manipulation der Stellgrößen erfolgt aufgrund einer Messung des realen Verlaufs der Regelgröße und deren Abweichung vom Sollverlauf, dem *Regelfehler*. Dadurch ist eine Reaktion auf unbekannte externe Einflüsse, sogenannte *Störgrößen* , möglich. Würde die Manipulation ohne Kenntnis des realen Regelgrößenverlaufs erfolgen, so spräche man von einer *Steuerung*.

2.1.1 Begriffsdefinitionen

Im folgenden sollen in der Regelungstechnik häufig benützte Begriffe definiert und erläutert werden.

Strecke, Prozeß

Generell werden Strecke und Prozeß synonym verwendet. Der Begriff Strecke spiegelt die Sichtweise einer kausalen Wirkungskette wieder. Es wird also die in Abbildung 2.3 gezeigte Sichtweise – links der Eingriff durch die Stellgröße u, rechts der daraus resultierende Effekt, die Regelgröße y – übernommen. Im Begriff des Prozesses findet sich mehr der Ablaufvorgang eines technischen Verfahrens wieder.

Das genaue Verhalten eines Prozesses, also das Zusammenspiel der verschiedenen beteiligten physikalischen Größen wird mit Hilfen von *Blockdiagrammen* dargestellt. Abbildung 2.1 zeigt den prinzipiellen Aufbau eines der verwendeten Blöcke. Das Gesamtsystem ergibt sich

dann als Differentialgleichungssystem, das durch die in den Blöcken verwendeten *Übertragungsfunktionen* bestimmt wird.

Übertragungsfunktion

Ein Block mit zugehöriger Übertragungsfunktion G(s) wird auch als Übertragungsglied bezeichnet. Er symbolisiert den kausalen Zusammenhang zwischen einer Ursache und deren Wirkung. Die Richtung des Signalflusses wird dabei mit Pfeilen ausgedrückt. Einzelne Blöcke lassen sich zu einem Gesamtsystem kombinieren, indem Signale aufgeteilt bzw. durch Blöcke mit mehreren Eingängen kombiniert werden. Eine mögliche Kombination ist zum Beispiel die Addition zweier Signale. Die Basisoperationen Addition und Subtraktion werden in der Regel als Kreise mit entsprechendem Vorzeichen notiert.

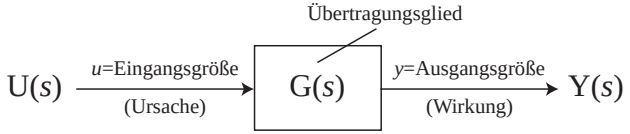


Abbildung 2.1: Die Blockdarstellung eines Übertragungsgliedes, nach Jörgl [48].

Normalerweise wird die Übertragungsfunktion in einer speziellen Notation, nämlich als *Laplace-Transformierte* angegeben. Dies hat den Vorteil, daß in Kombination mit der Blockschreibweise meist anhand von einfachen Rechenregeln das Verhalten des Gesamtsystems bestimmt werden kann. Die Laplace-Transformation $\mathfrak{L}$ entspricht einem Übergang vom *Zeitbereich* in den *Frequenzbereich*, und ist folgendermaßen definiert:

$$\mathfrak{L}[\mathrm{x}(t)] = \int_0^\infty \mathrm{x}(t)e^{-st}dt = \mathrm{X}(s) \tag{2.1}$$

Die Transformation wird normalerweise nicht explizit durchgeführt, da ein umfangreiches Tabellenwerk – auch *Korrespondenztabellen* genannt – mit allen gängigen Originalfunktionen und deren Tranformierten existiert. Wie in Gleichung 2.1 bereits angedeutet notiert man üblicherweise die *Zeitfunktion* x(t) in Kleinbuchstaben, die Laplace-Transformierte X(s) hingegen in Großbuchstaben.

Eine Übertragungsfunktion läßt sich auch anhand Ihrer Sprung-, Impuls- bzw. Anstiegsantwort charakterisieren. Bei der *Sprungantwort* handelt es sich um die Reaktion auf den sogenannten *Einheitssprung*, ein sprunghaftes Ansteigen des Eingabewertes von Null auf Eins mit anschließendem Halten des Wertes Eins. Die *Impulsantwort* ist die Reaktion auf die *Dirac-Funktion* $\delta(t)$ als Eingabe. Dabei handelt es sich um einen infinitesimal kurzen Impuls zum Zeitpunkt $t = 0$ mit der Eigenschaft $\int \delta(t)dt \equiv 1$. Die *Anstiegsantwort* ist das Verhalten bei Präsentation einer Eingabe mit konstantem Anstieg und Steigung Eins. Die Sprungantwort des später besprochenen PID-Reglers ist z.B. in Abbildung 2.4 gezeigt. Es ist üblich, einen Block mit einer Grafik der Sprungantwort zu versehen, um auf die enthaltene Charakteristik hinzuweisen.

Zu einer umfassenden Einführung in diese Thematik ist die Lektüre von [29] oder [57] zu empfehlen. An dieser Stelle soll nur grob auf die verschiedenen Fälle von Übertragungsfunktionen eingegangen werden.

Linearer Fall Der Begriff *linear* bezieht sich hier auf das zugrunde liegende Differentialgleichungssystem. Somit hat also die Übertragungsfunktion die folgende Form:

$$b_0 y + b_1 \dot{y} + b_2 \ddot{y} + \ldots + b_n y^{(n)} = a_0 u + a_1 \dot{u} + a_2 \ddot{u} + \ldots + a_m u^{(m)}$$

Im Frequenzraum entspricht dies bei in der Regel vorrausgesetzten „verschwindenden" Anfangsbedingungen $\lim_{t\to 0+} \mathrm{y}(t) = 0$ und $\lim_{t\to 0+} \mathrm{u}(t) = 0$ der Gleichung:

$$\mathrm{Y}(s) = \mathrm{G}(s)\mathrm{U}(s) = \frac{a_0 + a_1 s + a_2 s^2 + \ldots + a_m s^m}{b_0 + b_1 s + b_2 s^2 + \ldots + b_n s^n}\mathrm{U}(s) \tag{2.2}$$

Meist wird die Übertragungsfunktion als Bruch mit *Zähler-* und *Nennerpolynom* angegeben, wie es auf der rechten Seite von Gleichung (2.2) gezeigt ist. Man erkennt leicht, daß durch Kombination verschiedener linearer Übertragungsfunktionen wieder eine lineare Übertragungsfunktion entsteht. So ergibt etwa die Parallel- bzw. Reihenschaltung von k Übertragungsfunktionen $\sum_{i=1}^{k} \mathrm{G}_i(s)$ bzw. $\prod_{i=1}^{k} \mathrm{G}_i(s)$ als neue Übertragungsfunktion. Aufgrund der guten mathematischen Behandelbarkeit von linearen Differentialgleichungssystemen sind in der Regelungtechnik im Laufe der Zeit umfangreiche Werkzeuge zur Modellierung und Analyse solcher Systeme entstanden.

Nichtlinear Fall Im Gegensatz zum linearen Fall können nichtlineare Übertragungsfunktionen , wie sie in Abbildung 2.2 zu sehen sind, nicht ohne weiteres behandelt werden. Einfache statische Nichtlinearitäten sind bei genauer Kenntnis des Verlaufes oft unkritisch und können modelliert werden. Bei komplexeren Zusammenhängen, wie etwa Reibungseffekten, behilft man sich oft mit einer Linearisierung durch die Taylorreihenentwicklung. Steht keine vernünftige Differentialgleichung zur Verfügung, etwa weil die zugrunde liegenden physikalischen Prozesse nur ungenügend bekannt sind, so kann häufig durch Messungen ein einfaches Kennfeld zur Modellierung hinterlegt werden. Hochkomplexe nichtlineare Zusammenhänge wie Hysteresen werden ausschließlich über Spezialverfahren [50, 60] modelliert.

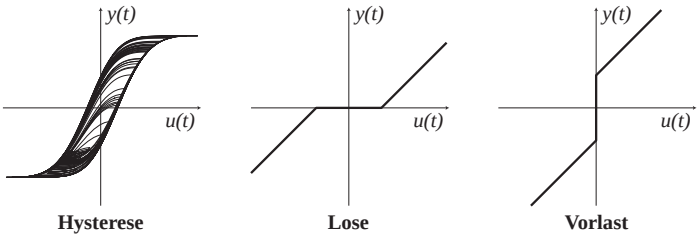


Abbildung 2.2: Drei typische nichtlineare Übertragungsfunktionen : Hystereseeffekte findet man typischerweise bei elektromagnetischen Vorgängen. Die Lose ist häufig in mechanischen Systemen – zum Beispiel als Getriebespiel – vertreten. Vorlast ist ein Effekt der sich bei Coulomb-Reibung – speziell beim Übergang von Haft- zu Gleitreibung – beobachten läßt.

Im nichtlinearen Umfeld finden sich die typischen Aufgaben der Modellbildung wieder, wie sie in Kapitel 3 noch beschrieben werden.

Zustandsraum

Die Darstellung dynamischer Systeme in der Regelungstechnik ist nicht nur auf den Frequenzraum beschränkt. Als alternative Beschreibungsform für lineare zeitinvariante dynamische Systeme gibt es die Zustandsraumbeschreibung im Zeitbereich. Sie besteht aus der *Zustandsgleichung* für den *Zustand* $\mathrm{x}(t)$ und der *Ausgangsgleichung* für die *Ausgangsgröße* $\mathrm{y}(t)$:

$$\begin{aligned} \dot{\vec{x}} &= \mathbf{A}\vec{x} + \mathbf{B}\vec{u} + \mathbf{E}\vec{z} \quad \text{mit} \quad \mathrm{x}(t_0) = \vec{x}_0 \\ \dot{\vec{y}} &= \mathbf{C}\vec{x} + \mathbf{D}\vec{u} + \mathbf{F}\vec{z} \end{aligned} \tag{2.3}$$

Neu hinzugekommen ist bei dieser Darstellung die unbekannte *Störgröße* $\mathrm{z}(t)$. Diese wird in der Zustandsgleichung unmittelbar berücksichtigt und nicht, wie in der Übertragungsfunktionsnotation, getrennt aufgeschaltet (siehe Abbildung 2.3). Da sowohl durch Übertragungsfunktionen als auch durch die Zustandsraumbeschreibung lineare zeitinvariante dynamische Systeme modelliert werden, können beide Varianten ineinander überführt werden. Die Sichtweise eines Zustandraumes entspricht allerdings mehr dem Modellierungsgedanken neuronaler Netze, die ebenfalls über Zustände, allerdings auch über nichtlineare Aktivierungsfunktionen, verfügen. Betrachtet man das Simulationsverfahren rekurrenter Netze (siehe hierzu auch Abschnitt 3.3.2), so stellt man fest daß es sich um ein diskretes Verfahren zum Lösen von Differentialgleichungsystemen – das Euler-Verfahren – handelt.

Regler

Wie bereits angedeutet, besteht die Aufgabe eines Reglers darin, das Verhalten eines geschlossenen Regelkreises – wie in Abbildung 2.3 gezeigt – in einer bestimmten Art und Weise zu beeinflussen. Das kann das Nachfahren eines vorgegebenes dynamischen Profils sein, aber auch das exakte Halten eines fixen Wertes. Dazu stehen dem Regler normalerweise verschiedene Signale zur Verfügung:

Die sogenannte *Führungsgröße* $\mathrm{W}(s)$ gibt den gewünschten Verlauf der *Regelgröße* $\mathrm{Y}(s)$ vor. Bei der *Regeldifferenz* $\mathrm{E}(s)$ handelt es sich um die Abweichung der Regelgröße von der Führungsgröße. Das vom Regler generierte Signal zur Beeinflussung des Prozesses ist die *Stellgröße* $\mathrm{U}(s)$.

PID-Regler Der sogenannte **P**roportionalwirkende, **I**ntegrierende und **D**ifferenzierende Regler ist wohl einer der am häufigsten im Einsatz befindlichen Regler überhaupt. Genaugenommen handelt es sich um die Kombination der drei eben genannten Anteile P, I und D. Somit können durch Verzicht auf einzelne Anteile verschiedene Variationen – je nach Anforderung der Regelstrecke – genutzt werden. Das zugrunde liegende Differentialgleichungsystem bzw. die Übertragungsfunktion $\mathrm{G_R}(s)$ im idealen Fall[1] lauten:

$$u = K\left[e + \frac{1}{T_n}\int e + T_v\dot{e}\right] \quad \equiv \quad \mathrm{G_R}(s) = K\frac{1 + T_n s + T_n T_v s^2}{T_n s}$$

Dabei gibt K den proportionalen, $\frac{K}{T_n}$ den integralen und KT_v den differentiellen Verstärkungsfaktor vor. Der proportionale Anteil sorgt für Ausregeln der Regeldifferenz gegen Null,

[1] Ideale PID Regler sind nicht realisierbar, da das Differenzieren aus verschiedenen Gründen, wie etwa Rauschen, Probleme bereitet. Stattdessen kommen PID-Regler mit einem DT1-Glied (Vorhalteglied) als Ersatz für den D-Anteil zum Einsatz. Näheres hierzu findet sich z.B. in [57].

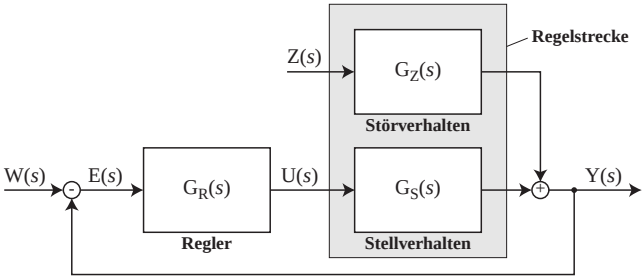


Abbildung 2.3: Das Standardblockschaltbild des Regelkreises, nach Jörgl [48]. Man erkennt die Rückkopplung des Ausgangs $Y(s)$ und den Vergleich mit der Sollvorgabe $W(s)$. Darüberhinaus sind aber auch Regler im Einsatz, die nicht mit dem resultierenden Fehler $E(s)$, sondern direkt mit $W(s)$ und $Y(s)$ versorgt werden. Wichtig ist der Einfluß des Störverhaltens $G_Z(s)$, das unbekannt ist. Denn wäre das Stellverhalten $G_S(s)$ bekannt und kein Störverhalten vorhanden, so könnte ohne Rückkopplung gearbeitet werden. Man erhielte dann eine Steuerung.

wenn kein fester Anteil der Stellgröße $\neq$ Null nötig ist. Der integrale Anteil kann genau diese Offset-Fehler (Sprünge) kompensieren. Dabei gibt die Nachstellzeit T_n diejenige Zeit an, die benötigt wird, bis die Stellgröße bei der Sprungantwort denselben Wert erreicht, der durch Kombination mit dem Proportionalanteil sofort eingestellt wird. Über den differentiellen Anteil schließlich läßt sich das Verhalten bei Anstiegsantworten beschleunigen. Die Vorstellzeit T_v gibt dabei an, wielange der Proportionalanteil bräuchte, um dieselbe Stellgrößenänderung bei der Anstiegsantwort zu realisieren. Offensichtlich ergänzen sich alle drei Anteile so, daß insgesamt ein schnelleres Regelverhalten entsteht als durch Verwendung der einzelnen Komponenten möglich wäre. Dies ist im Allgemeinen auch das Ziel einer Regelung: Möglichst schnell auf eine Störung zu reagieren und die Regeldifferenz gegen Null gehen zu lassen. Die Abbildung 2.4 zeigt das Verhalten des PID-Reglers beim Einheitssprung bzw. der Identität als Eingabe.

Stabilität

Eines der wichtigsten Kriterien zur Beurteilung eines Regelkreises ist seine Stabilität. Aufgrund von Zeitverzögerungen in Kombination mit Signalverstärkungen kann es im geschlossenen Regelkreis nämlich zu Schwingungen mit steigender Frequenz und Amplitude oder einem unbeschränkten Ansteigen der Regelgröße $Y(s)$ kommen. Das Gesamtsystem wird dann als instabil bezeichnet. Generell gilt ein lineares zeitinvariantes dynamisches System als stabil, wenn es auf ein beschränkte Eingangsgröße stets mit einer beschränkten Ausgangsgröße reagiert[2]. Für gebrochen rationale Übertragungsfunktionen kann Stabilität anhand der Nullstellen des Nennerpolynoms, den *Polstellen*, überprüft werden. Hier sei noch kurz auf sogenannte verdeckte Pole hingewiesen. Das bedeutet, daß eine Nullstelle des Nennerpolynoms ebenso als Nullstelle des Zählerpolynoms vorkommt, und sich folglich kürzen läßt. Dennoch existiert diese Polstelle im realen System und führt eventuell zu unvorhersehbarem, instabilem Verhalten. Dies ist vor allem dann relevant, wenn kein Modell des Systems existiert und man auf Messungen zur Modellierung des realen Systems angewiesen ist!

[2]In der Praxis wird diese Forderung meist noch weiter aufgeweicht, so daß Stabilität nur bezüglich aller realistisch möglichen Eingaben gefordert wird.

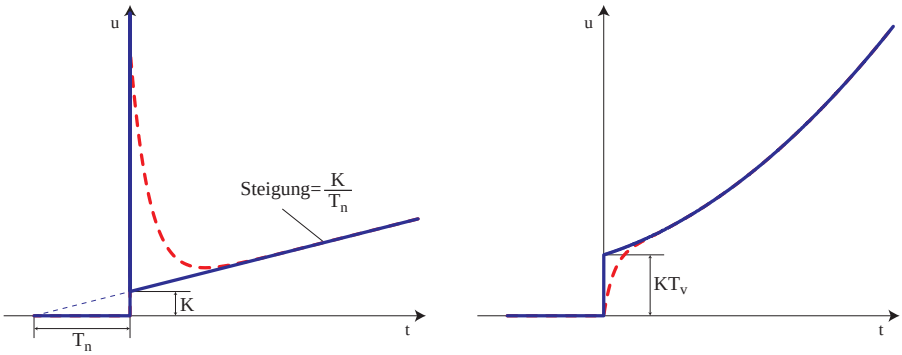


Abbildung 2.4: Links ist die Sprungantwort und rechts die Anstiegsantwort eines PID-Reglers zu sehen. Sie werden durch die Wahl der Parameter K, T_n und T_v bestimmt. Bei der Sprungantwort gibt der Wert von K den Offset vom Nullpunkt, verursacht durch den Proportionalanteil vor. Das Verhältnis $\frac{K}{T_n}$ bestimmt die Steigung durch den Integralanteil. Zusätzlich zum blau durchgezogen gezeigten *idealen* PID-Regler ist rot gestrichelt die *Realisierung* mit DT1-Glied dargestellt.

Darüberhinaus existieren weitere Kriterien, wie etwa das von Hurwitz oder Nyquist , um Stabilitätsaussagen machen zu können. Auch für Systeme die nur teilweise nichtlinear sind, können diese Kriterien – nach Linearisierung – genutzt werden. Es ist jedoch zu beachten, daß durch die Linearisierung genau die instabile Charakteristik verschwinden kann!

2.2 Probleme bei der Regelung

Trotz der umfangreichen Theorie im Feld der Regelungstechnik existieren – speziell bei nichtlinearen zeitvarianten dynamischen Systemen – immer noch Probleme bei der Regelung, die im folgenden kurz beschrieben werden sollen.

Nichtlinearitäten. Effekte, wie die in Abbildung 2.2 gezeigte Lose, Hysterese oder Vorlast können durch Sprung- bzw. Unstetigkeitsstellen in Kombination mit klassischen Reglern zu ungewollten Schwingungen oder im ungünstigsten Fall zum Stabilitätsverlust des geschlossenen Regelkreises führen.

Schleichende Parameteränderungen. Wie erwähnt, stehen zur Justierung des oben vorgestellten PID-Reglers mehrere Parameter zur Verfügung. Diese werden so eingestellt, daß *in Kombination* mit der zu regelnden Strecke ein Kompromiß zwischen Stabilität und Ansprechgeschwindigkeit des geschlossenen Regelkreises erreicht wird. Ändert sich das Verhalten der Regelstrecke, zum Beispiel durch Verschleiß eines Lagers oder äußere Einflüsse wie Temperatur, so ist die gewählte Einstellung nicht mehr optimal. Dies bedeutet eine Verschlechterung der Stabilität oder des Ansprechverhaltens.

Unsichtbare Zustände. Die Verschattung einer Polstelle der Übertragungsfunktion kann für bestimmte Eingangssignale zur Instabilität des ansonsten stabilen Systems führen.

Totzeiten. Tritt der Effekt oder Teileffekt eines Regeleingriffes erst nach einer bestimmten Verzögerungszeit auf, so kann mit klassischen Reglern nur unter starken Einschränkungen, wie etwa sehr langsamem Ansprechverhalten, gearbeitet werden. Solche Effekte treten immer dann auf, wenn Transportprozesse beteiligt sind. So etwa bei Rohrleitungen oder Temperaturflußprozessen.

2.3 Modelle in der Regelung

Einige der eben genannten Schwierigkeiten – wie etwa eine bestimmte Klasse von Nichtlinearitäten oder Totzeiten – lassen sich durch den geschickten Einsatz von Prozeßmodellen umgehen. Vor dem Einsatz eines solchen Modells stellt sich allerdings die Frage, wie man das Modell erhält. Abbildung 2.5 zeigt die typische Vorgehensweise um ein Gesamtprozeßmodell zu generieren. Üblicherweise wird die Modellbildung Off-Line durchgeführt, das bedeutet man lernt eine zeitinvariante Abbildung des Prozesses. Dies führt zu einer Reihe von Einschränkungen der vorgenommenen Modellierung, weshalb im nächsten Kapitel kurz auf das Thema On-Line-Modellbildung eingegangen wird, um die Notwendigkeit eines solchen Verfahrens näher zu erläutern.

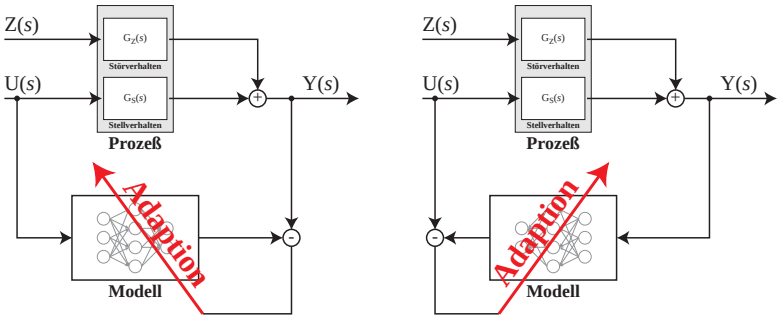


Abbildung 2.5: Die in der Regelung typischerweise eingesetzten Prozeßmodelle werden durch Parallelschalten einer Modellstruktur und anschließender Adaption identifiziert. Hierdurch wird normalerweise, wie im linken Teil zu sehen ist, die Abbildung $U(s) \rightarrow Y(s)$ gelernt. In einigen Fällen – zum Beispiel beim Einsatz zur Vorsteuerung – benötigt man jedoch die Umkehrabbildung $Y(s) \rightarrow U(s)$. Man lernt dazu ein Prozeßmodell der inversen Dynamik und erhält ein inverses Prozeßmodell, wie es im rechten Teil gezeigt ist. Normalerweise erfolgt die Identifikation Off-Line. Dies bereitet allerdings Probleme, sobald der Prozeß zeitvariant ist und Parameter On-Line nachadaptiert werden müßten.

2.3.1 Einsatz von Gesamtprozeßmodellen

Zum Einsatz von Prozeßmodellen in der Regelung existieren eine Reihe von Reglerstrukturen, in denen einerseits spezialisierte Regler Prozeßmodelle nutzen, andererseits inverse Prozeßmodelle direkt als Regler eingesetzt werden. Im folgenden sollen je eine dieser Techniken kurz erläutert werden:

Inverse Prozeßmodelle als Vorsteuerung. Hierbei wird, wie in Abbildung 2.6 zu sehen ist, das inverse Prozeßmodell direkt parallel zum Regler eingesetzt. Verfügt man über ein exaktes inverses Modell der Regelstrecke, so liefert dieses bereits die notwendige Stellgröße U(s) um den Verlauf von Y(s) der Führungsgröße W(s) folgen zu lassen. Der eingesetzte Regler hat also nur noch die Aufgabe, unvorhersehbare Störungen auszuregeln [106]. Eine einfache Form dieses Ansatzes findet man häufig als Kennfeldrealisierung, wobei anhand von Messungen das Verhalten der Regelstrecke grob genähert wird, um zum Beispiel einem PI-Regler eine bessere Dynamik zu verleihen. Als Nachteil dieser Vorgehensweise muß gewertet werden, daß ein schlechtes inverses Prozeßmodell die Reglerperformance deutlich verringern kann. Speziell Prozeßmodelle aus dem Bereich der neuronalen Netze neigen aber dazu, unvorhersehbare Werte zu liefern, wenn als Eingabe bisher nicht trainierte Werte präsentiert werden.

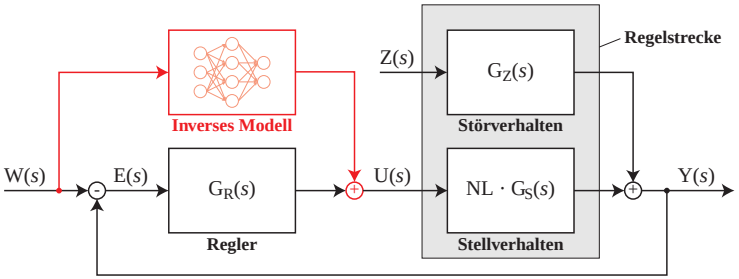


Abbildung 2.6: Geschlossener Regelkreis mit nichtlinearem Stellverhalten NL · $G_S(s)$ und einem inversen Prozeßmodell als Vorsteuerung zur Unterstützung eines konventionellen Reglers.

Regelung durch Modellvorhersagen. Diese Art von Regelung nutzt ein Prozeßmodell, um in jedem Regelungsschritt für einen festgelegten Zeithorizont eine Ziel- bzw. Kostenfunktion J zu minimieren [32]. Anhand des Ergebnisses kann – wie es Abbildung 2.7 zeigt – ein Regler adaptiert werden oder sogar aus einer Menge von Reglertypen der optimale gewählt werden. Der Zeithorizont gibt dabei vor, bis zu welchem Zeitpunkt das *geschätzte* zukünftige Prozeßverhalten betrachtet wird, um die aktuelle Regelungsstrategie festzulegen.

Eine typische Zielfunktion ist zum Beispiel das Integral bzw. die Summe über die Quadrate der *zu erwartenden* Regeldifferenz, die der Differenz der Führungsgröße und der *geschätzten* Regelgröße entspricht. Häufig findet man noch einen Strafterm, der verhindern soll, daß allzu abrupte Regeleingriffe erfolgen, und erhält nach [15] somit:

$$\mathrm{J}(N_1, N_2, N_u) = E\left[\sum_{i=N_1}^{N_2} \delta(i)(\mathrm{y}(t+i|t) - \mathrm{w}(t+i))^2 + \sum_{j=1}^{N_u} \lambda(j)\Delta\mathrm{u}(t+j-1)^2\right]$$

Dabei gibt N_u vor, bis zu welchem Zeitpunkt glatte Regeleingriffe vorrausgesetzt werden. Die Werte von N_1 und N_2 definieren den Zeithorizont, in dem die Regelung korrekt sein soll. Somit geht die zu erwartende Regeldifferenz nur innerhalb dieses Zeitfensters in die Kostenfunktion ein. Das bedeutet für hohe Werte von N_1, daß die unmittelbare

Reaktion auf die Änderung der Stellgröße u nicht wichtig erscheint, aber längerfristig die Regeldifferenz gegen Null gehen soll. Enthält die Regelstrecke zum Beispiel eine Totzeit, so kann N_1 größer oder gleich dieser Totzeit gewählt werden, da Regeleingriffe frühestens nach Ablauf dieser Totzeit Auswirkungen haben. Die Gewichtungsfaktoren $\delta(i)$ und $\lambda(j)$ können dazu benützt werden, eine zeitliche Gewichtung der Zielfunktion vorzunehmen. Typischerweise nutzt man eine exponentiell abfallende Funktion, so daß weit in der Zukunft liegende Regeldifferenzen schwächer eingehen. Die Schreibweise $y(t+i|t)$ deutet an, daß zukünftige Werte $t+i$ auf der Basis des Wissens zum Zeitpunkt t geschätzt werden. Der Nachteil dieses Ansatzes ist einerseits der hohe Simulationsaufwand, da pro Vorhersage eine komplette zukünftige Prozeßtrajektorie simuliert werden muß. Andererseits benötigt man für diese Simulation sehr genaue Modelle, da eine Nachführung der Simulation einer *zukünftigen* Prozeßtrajektorie durch Messungen am laufenden Prozeß offensichtlich nicht möglich ist.

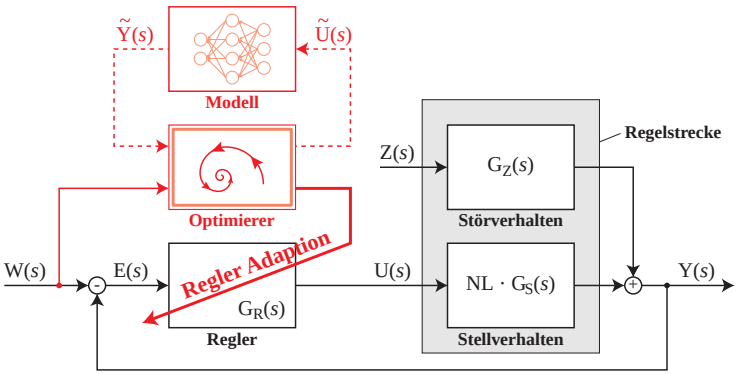


Abbildung 2.7: Der Systemaufbau zur Regelung mit Hilfe von Modellvorhersagen.

Die eben erläuterte Strukturen sollen nur ein Gefühl für die Möglichkeiten der modellgestützten Regelung geben. Ansonsten sei auf die reichhaltige Literatur zu diesem Thema verwiesen. Unter anderem sind modellgestützte Ansätze zu finden in [1, 69, 68, 61].

2.3.2 Neutralisierung einzelner Nichtlinearitäten

Neben dem Einsatz von Gesamtprozeßmodellen zur Regelung nichtlinearer Prozesse kann mit Hilfe von Teilmodellen eine vorhandene Nichtlinearität neutralisiert werden. Man erhält so eine mit klassischen Reglern beherrschbare lineare Regelstrecke. Voraussetzung für diese Vorgehensweise sind einerseits eine ausreichende Kenntnis der Regelstrecke, andererseits die Möglichkeit, die Nichtlinearität zu isolieren. Damit ist gemeint, daß sowohl Messungen zur Identifikation als auch Möglichkeiten zur korrekten Anbindung des erhaltenen Modells bestehen. Neuere Arbeiten aus dem Gebiet der Neuronalen Netze zeigen hier auch Wege auf, um nicht meßbare Nichtlinearitäten zu identifizieren [9].

Sind obige Voraussetzungen erfüllt, so können mit Hilfe eines inversen Modells Hintereinanderschaltungen von nichtlinearen und linearen Anteilen linearisiert werden. Die Kopplung der

Nichtlinearität mit dem inversen Modell ergibt offensichtlich die Identitätsabbildung, die in Abbildung 2.8 grau unterlegt ist . Somit bleibt nur der lineare Anteil des Teilprozesses erhalten.

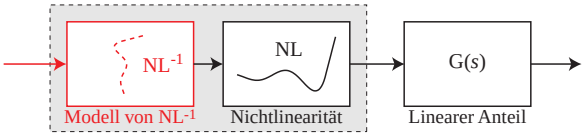


Abbildung 2.8: Durch Vorschalten eines inversen Modells der Nichtlinearität erscheint die Gesamtstrecke linear. Dies entspricht einer Vorsteuerung und wird häufig durch Kennfelder realisiert.

Additive Verschaltungen nichtlinearer und linearer Teilprozesse wie in Abbildung 2.9 können durch den Einsatz von direkten Modellen linearisiert werden. Die Subtraktion der Modellausgabe neutralisiert dabei den Anteil des nichtlinearen Teilprozesses . Wiederum erhält man einen linearen Teilprozeß, der mit klassischen Methoden behandelt werden kann.

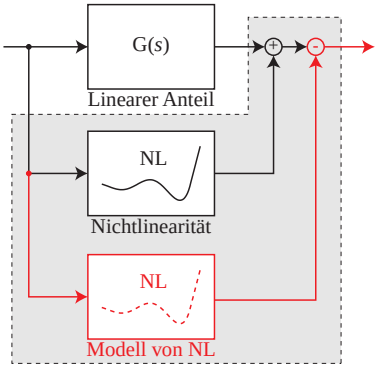


Abbildung 2.9: Durch subtraktives Aufschalten eines Modells der Nichtlinearität kann diese neutralisiert werden. Die Teilstrecke verhält sich dann linear und kann häufig mit klassischen Methoden geregelt werden.

Abschließend bleibt zu bemerken, daß häufig nichtlineare Effekte mit nur schwachen Auswirkungen auftreten, so daß eine explizite Behandlung unnötig ist. Summieren sich diese jedoch auf, so kann es wieder zu den genannten unerwünschten Effekten kommen. Eine allgemeingültige Vorgehensweise im Umgang mit nichtlinearen Einflüssen kann also nicht angegeben werden. Vielmehr muß jedesmal aufs neue die Charakteristik der Regelstrecke genau erfaßt und beurteilt werden.

Nach dieser allgemeinen Betrachtung von Modellen und ihrem Einsatz in der Regelungstechnik soll im nächsten Kapitel der Schwerpunkt auf den eigentlichen Modellbildungsprozeß gelegt werden.

Kapitel 3

Modellbildung

Das Wort „Modell“ hat auch im technischen Umfeld eine Vielzahl von Bedeutungen. So erstellt man zum Beispiel vor der eigentlichen Realisierung einer Idee ein Modell, um die prinzipielle Funktionsweise bereits vorab zu testen. Dies kann eine Simulation im Computer, aber auch die Anfertigung einer richtigen Miniatur sein. Häufig findet man aber auch den anderen, für uns interessanten Fall: Die Nachbildung eines bereits real existierenden Systems und deren Simulation im Computer. Eine derartige Modellvorstellung leitet sich direkt aus der Physik ab, die schon seit jeher versucht mit mathematischen Modellen unsere Umwelt zu erklären. Dieses Kapitel soll, eine einheitliche Sichtweise der dazu bekannten verschiedenen Verfahren vermitteln.

3.1 Der Modellbegriff

Ein Modell im Sinne unserer Problematik soll eine möglichst gute Approximation eines technischen Prozesses darstellen. Simulationen am Modell sollen dasselbe Verhalten zeigen, wie es durch Messungen am realen Prozeß in Erfahrung gebracht werden kann.

Zu Beginn des Modellierungsvorgangs muß die notwendige Genauigkeit, die sogenannte Granularität des Modells, bestimmt werden. Ist diese festgelegt, so kommt man zum eigentlichen Problem der Modellierung, nämlich die passende Modellstruktur zu finden. Ist diese gefunden, so muß noch eine Parameteranpassung – meist anhand von Meßdaten – vorgenommen werden. Eine der offensichtlichen Regeln lautet hierbei, so viel Vorwissen wie möglich einzubringen, um möglichst wenig Parameter justieren zu müssen. Betrachtet man diese Vorgehensweise, so lassen sich folgende Modelltypen unterscheiden:

White-Box-Modelle. Diese Modellklasse enthält alle bis ins Detail bekannten Modelle. Das bedeutet, daß man sowohl die physikalische Struktur als auch die notwendigen Parameter genauestens in Erfahrung bringen konnte. Das resultierende physikalische Modell ist also ein Kopie der Realität.

Grey-Box-Modelle. Hier besteht meist keine komplette Einsicht in komplexe physikalische Systeme. Dennoch kann in vielen Fällen die physikalische Struktur ganz oder teilweise erfaßt werden. Man unterscheidet wiederum zwei Arten:

1. Die physikalische Modellstruktur ist bekannt, eine Reihe von Parametern muß aber aus gemessenen Daten geschätzt werden. Dies ist ein extrem häufiger Fall.

2. Die physikalische Modellstruktur ist teilweise bekannt und die unbekannten Bereiche werden durch *Black-Box-Modelle* ersetzt.

Black-Box-Modelle. Ist man nicht in der Lage das zugrundeliegende physikalische Phänomen zu erkennen, so bedient man sich dieser Modellklasse. Meist kommen frei parametrierbare Modellstrukturen mit bekanntem Verhalten – wie etwa Polynome, neuronale Netze oder nummerische Modelle – zum Einsatz, um das reale Verhalten nachzubilden.

Die im folgenden behandelten Modellstrukturen gehören zur dritten Klasse, den Black-Box-Modellen. Obige allgemeine Sichtweise muß natürlich noch weiter konkretisiert werden, um ein sinnvolle Anwendung zu erlauben. Im folgenden sollen deshalb zwei Arten von Modellen unterschieden werden[1]:

Allgemeine Modelle. Diese Modelle stellen die allgemeinste Sichtweise dar: Sie können einen inneren Zustand $\vec{x}$ und einen Parametersatz $\vec{p}$ enthalten. Die Abbildungsfunktion lautet dann:

$$\mathrm{f}_{\vec{p}} : \{\vec{u}, \vec{x}_i\} \overset{\vec{p}}{\mapsto} \{\vec{y}, \vec{x}_{i+1}\} \tag{3.1}$$

Diese Art von Modellen ist also in der Lage sowohl statische als auch – mit Hilfe des inneren Zustands – dynamische Prozesse nachzubilden. Durch Modifikation des Parametervektors $\vec{p}$ ist auch eine Adaption möglich.

Differenzierbare Modelle. Als Spezialfall der „Allgemeinen Modelle“ ergibt sich eine Klasse von Modellen, für die die erste Ableitung von $\mathrm{f}_{\vec{p}}$ bezüglich des Parametervektors $\vec{p}$, auch Jacobi-Matrix genannt, angegeben werden kann. Mit dieser und einer beliebigen *differenzierbaren* Fehlerfunktion $\mathrm{E} : \{\vec{y}_{\text{Modell}}, \vec{y}_{\text{Prozeß}}\} \rightarrow \mathbb{R}^+$ zur Bewertung der Genauigkeit der bisherigen Approximation, können bekannte Minimierungsverfahren wie etwa Gradientenabstieg realisiert werden. Wichtig ist es hierbei, die Auswirkung des inneren Zustandes $\vec{x}$ auf die Jacobi-Matrix zu berücksichtigen, da es dadurch zu unerwünschten Nebeneffekten kommen kann!

Diese Unterscheidung hat zur Konsequenz, daß viele der Standardmodelle auf eine einheitliche Form des Lernens zurückgeführt werden können, die allgemein für beliebige Fehlerfunktionen implementiert werden kann.

3.2 Lernverfahren

Der Begriff des Lernens bzw. der Adaption wird hier im Sinne des „Lernens aus Beispielen“ verwendet. Ziel ist es dabei, eine unbekannte – auch dynamische, also gedächnisbehaftete – Funktion $\tilde{\mathrm{f}} : \vec{u} \mapsto \vec{y}$ anhand von gemessenen Datenpaaren $\{\{\vec{u}_1, \vec{y}_1\}, \ldots \{\vec{u}_n, \vec{y}_n\}\}$ möglichst genau zu approximieren.

[1] Eine Implementierung dieser Sichtweise findet sich in der Modellbibliothek AMoC [100] in Form von C++ Klassen. Eine Reihe von Standardmodellen und Optimierungsverfahren stehen dort bereits zur Verfügung.

3.2.1 Das Bias-Variance-Problem

Die zur Verfügung stehenden Datenpaare, haben – oft durch Meßrauschen bedingt – einen unbekannten Fehler. Desweiteren ist bei Verwendung eines Black-Box-Modells die Eignung der Struktur zur Approximation dieses Problems nicht genau bekannt und induziert damit unter Umständen einen inhärenten Fehler. Geman, Bienenstock, und Doursat [33] haben dieses Verhalten aus statistischer Sicht erstmals im Kontext der neuronalen Netze untersucht. Folgt man Wolpert [112], so läßt sich der Erwartungswert[2] des normalerweise verwendeten Fehlerquadrat-Abstandes folgendermaßen zerlegen:

$$\mathrm{E}(C|f,m,q) = \underbrace{\sigma^2_{f,m,q}}_{\text{fix}} + \underbrace{\mathrm{bias}^2_{f,m,q} + \mathrm{variance}_{f,m,q}}_{\text{reduzierbar}} \tag{3.2}$$

Wobei f für die zu approximierende Funktion, m für die Zahl der zur Verfügung stehenden Lerndaten und q für die verwendeten Testdaten steht. Untersucht man die Terme der rechten Seite näher, so stellt sich folgende Bedeutung heraus:

$\sigma^2_{f,m,q}$ Dies ist der von f durch Meßrauschen und ähnliche Effekte verursachte fixe – also nicht reduzierbare – Fehler. An dieser Stelle hilft also auch kein schlauer Lernalgorithmus, da keine Abhängigkeit zu den gegebenen Lerndaten vorhanden ist. Der Term kann allerdings minimiert werden, indem Vorwissen über die Art des Rauschens oder mögliche Meßfehler bereits in der Datenvorverarbeitung berücksichtigt werden.

$\mathrm{bias}^2_{f,m,q}$ Der *Bias* ist ein Maß für die zu erwartende Genauigkeit der Approximation bei den Testdaten q. Hierauf kann durch Variation der Lerndauer, der Modellkomplexität und des Lernverfahrens stark Einfluß genommen werden.

$\mathrm{variance}_{f,m,q}$ Die *Variance* gibt an, wie stark der zu erwartende Fehler bei Nutzung unterschiedlicher Lerndatensätze schwankt. Meist hängt diese Größe hauptsächlich von der Modellkomplexität und Lerndauer, weniger hingegen vom Lernverfahren ab.

Man könnte nun auf die Idee kommen, primär den *Bias*-Term zu reduzieren, da eine möglichst geringe Abweichung von der Originalfunktion das Ziel einer Approximation ist. Unglücklicherweise führt dies in der Regel zu einer größeren *Variance*: Erhöht man die Zahl der freien Parameter, so kann die Trainingsmenge besser gelernt werden. Da das Rauschen mitgelernt wird, führt dies zu einer ungewollt großen Varianz. Senkt man die Zahl der Parameter, so kann der Originalverlauf nicht mehr genügend approximiert werden, der *Bias* steigt. Die *Variance* hingegen sinkt, da das Rauschen nicht mehr signifikant gelernt wird. Ähnliche Effekte lassen sich auch durch Variation der Lerndauer erzielen. Abbildung 3.1 zeigt nochmals an einem Beispiel, wie sich die geschilderte Problematik auswirkt.

Besitzt man nur sehr wenige Daten, so ist dieses Problem nicht lösbar, da nicht klar ist, welcher Anteil des Signals Rauschen darstellt und welcher Anteil dem ursprünglichen Signal zuzuschreiben ist. Dennoch existieren einige Ansätze, die versuchen das Problem zu lindern. Die beiden am weitesten verbreiteten Techniken sollen nun kurz angeschnitten werden.

[2] Die Mittelung erfolgt über verschiedene Sets von Lerndaten gleicher Größe m.

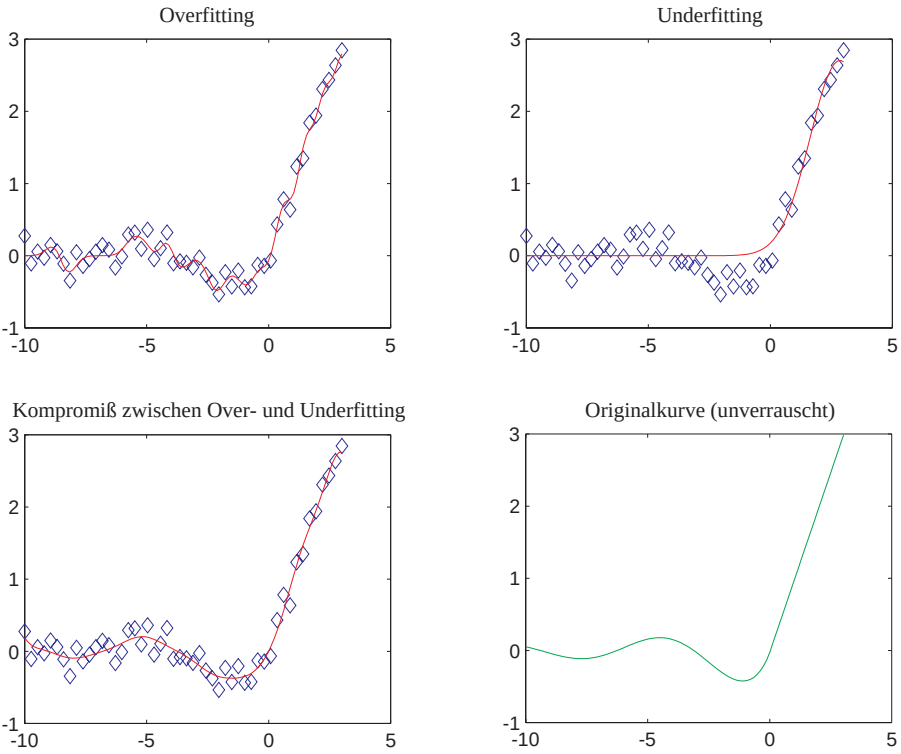


Abbildung 3.1: Ein Beispiel für das Bias-Variance-Problem: Die Originalkurve rechts unten wurde durch Rauschen verfälscht und verschiedenen Modellen zum Lernen angeboten. Die verfälschten Meßpunkte sind als blaue Rauten dargestellt. Die linke obere Grafik stellt „Overfitting“ bzw. Überlernen dar. Hier wurde auch das Rauschen mitgelernt, wodurch – wenn man die Originalkurve kennt – unsinnige Ausgaben erzeugt werden. Rechts daneben ist „Underfitting“ bzw. Unterlernen gezeigt. Der Verlauf der Originalkurve ist nur grob approximiert. Links neben der Originalkurve ist deutlich erkennbar, daß ein guter Kompromiß zwischen Über- und Unterlernen existiert. Dennoch wäre ohne Kenntnis der Originalkurve nicht klar, welche der drei Approximationen die korrekte ist!

Modellauswahlverfahren

Unter dem Begriff „Model Selection“ existieren eine Reihe von Verfahren, die größtenteils aus dem Bereich der neuronalen Netze stammen. Üblicherweise wird der Generalisierungsfehler[3] verschiedener Modelle, bzw. desselben Modells mit variierender Parameterzahl, anhand einer Validierungsmenge geschätzt und das Modell mit dem niedrigsten Fehler wird gewählt.

Im Zusammenhang mit der Schätzung dieses Fehlers kommen verschiedene Techniken zum Einsatz. Einerseits ersetzen häufig statistische Maße wie etwa „Schwarz's Bayesian Criterion“ [91] oder „Generalized Prediction Error“ [63] die Validierungsmenge, was bei komplexeren Modellen häufig zu Überlernen führt.

Andererseits können über sogenannte „Resampling“-Techniken wie etwa „Crossvalidation“ oder „Bootstrapping“ [22, 23, 24], die zur Verfügung stehenden Trainingsdaten quasi optimal zur Schätzung des Generalisierungsfehlers genutzt werden. Aus der zur Verfügung stehenden Datenmenge werden mehrere Submengen gewonnen, die dann wechselweise trainiert und validiert werden, um eine Schätzung des Generalisierungsfehlers zu erhalten.

Ebenfalls zur Klasse der Modellauswahl gehören Verfahren die die Struktur des Modells anhand des Generalisierungsfehlers modifizieren, um niedrig komplexe Modelle zu erhalten. Beim „Pruning“ [78] neuronaler Netze etwa wird versucht mit Hilfe der zweiten Ableitung „unwichtige“ Parameter oder sogar komplette Neuronen aus dem Netz zu entfernen, um ein Netz mit möglichst geringer Komplexität zu erhalten. Hierzu gehören die auf Seite 34 beschriebenen Techniken „Optimal Brain Damage“ und „Optimal Brain Surgeon“.

Regularisierung

Hier wird im Gegensatz zur oben erläuterten Vorgehensweise die Menge der möglichen Lösungen *eines* Modells beschränkt. Dies kann zum Beispiel durch einen zusätzlichen Strafterm im Trainingsverfahren [103] oder durch vorzeitiges Abbrechen des Trainings erfolgen:

Strafterm basierte Verfahren Meist wird mit Hilfe des Strafterms eine zusätzliche Eigenschaft, wie etwa „Glattheit“, der zu approximierenden Funktion eingeführt. Dies erfordert Vorwissen diesbezüglich und führt zu meist sehr speziellen Straftermen. Ein bekannter Ansatz hierzu ist „Weight-Decay“ aus dem Umfeld der neuronalen Netze. (Siehe Seite 33)

Early Stopping Verfügt man über eine ausreichende Datenmenge, um diese in eine separate Trainings-, Test- und Validierungsmenge aufteilen zu können, so empfiehlt sich das im folgenden beschriebene Verfahren, um die Lerndauer festzulegen:

Das Modell wird mit Hilfe der Trainingsmenge wiederholt trainiert. Nach jedem Trainingsdurchlauf wird der Fehler auf der Validierungsmenge überprüft. Solange dieser Fehler kleiner wird, wird mit der Trainingsmenge weitertrainiert, ansonsten wird abgebrochen. So wird die Lerndauer beschränkt und ein Überlernen in den meisten Fällen vermieden. Die Testmenge dient im Anschluß daran dazu, den Generalisierungsfehler festzustellen.

[3] Mit Generalisierung ist die Fähigkeit gemeint, von Lerndaten zu abstrahieren und auf den zugrunde liegenden Generierungsprozeß zu schließen. Somit können auch bisher ungesehene Daten dieses Prozesses korrekt modelliert werden.

Die Kombination mehrerer der genannten Techniken führt zur Netzarchitektur „FMS“ , die unter der Prämisse „Netze geringer Komplexität“ entstanden ist. (Siehe Seite 33)

Ein Vergleich verschiedener Varianten, die sich im wesentlichen in der Berechnung des Generalisierungsfehlers – wie oben beschrieben – unterscheiden, ist in [85] zu finden. Eine erschöpfende Behandlung des gesamten Themenkomplexes findet sich auch in [40].

3.2.2 Belernen Allgemeiner Modelle

Das Lernen bei allgemeinen Modellen ist offensichtlich stark von der gewählten Modellstruktur abhängig und läßt sich nicht davon abstrahieren. Aus diesem Grund werden einige dieser Lernverfahren, wie etwa „Real Time Recurrent Learning“ (Seite 29) oder „Backpropagation Through Time“ (Seite 28) erst bei der Vorstellung der zugehörigen Modelle erläutert.

3.2.3 Belernen differenzierbarer Modelle

Das Lernen für diese speziellen Modelle läßt sich auf eine gemeinsame Form zurückführen. Die Fehlerfunktion[4] E wird dabei durch gradientenbasierte Verfahren minimiert. Um die bereits erwähnten Erweiterungen zur Linderung des Bias-Variance-Problems, wie etwa Weight-Decay, nutzen zu können wird zusätzlich zur eigentlichen Fehlerfunktion ein Strafterm $S : \vec{p} \mapsto \mathbb{R}^+$ eingeführt, der ebenfalls beliebig modifiziert werden kann.

Um für verschiedene Fehlerfunktionen und Strafterme nicht jeweils die komplette Gradientenberechnung neu implementieren zu müssen, wird folgender Zusammenhang genutzt:

$$\frac{\partial(\mathrm{E}(\mathrm{f}_{\vec{p}}(\vec{u}), \vec{y}_{\text{Prozeß}}) + \mathrm{S}(\vec{p}))}{\partial \vec{p}} = \frac{\partial \mathrm{E}(\vec{y}, \vec{y}_{\text{Prozeß}})}{\partial \vec{y}} \cdot \frac{\partial \mathrm{f}_{\vec{p}}(\vec{u})}{\partial \vec{p}} + \frac{\partial \mathrm{S}(\vec{p})}{\partial \vec{p}} \tag{3.3}$$

Die Ableitung der Fehlerfunktion in Kombination mit dem Strafterm zerfällt also in die Jacobi-Matrizen der Fehlerfunktion E, des Modells $\mathrm{f}_{\vec{p}}$ und des Strafterms S. Eine Modifikation der Fehlerfunktion oder des Strafterms führt also *nicht* zwangsläufig zu einer Reimplementierung eines gradientenbasierten Lernverfahren. Zu erwähnen bleibt, daß natürlich die Matrixmultiplikation der Preis für diesen Umstand ist.

Im weiteren Verlauf werden einige der bekannten gradientenbasierten Verfahren zur Funktionsminimierung besprochen.

Gradientenabstiegsverfahren - Backpropagation

Zu Beginn eine kurze Klärung des Begriffs „Backpropagation“ [83], der fälschlicherweise häufig synonym zu Gradientenabstieg verwendet wird.

Beim Backpropagation handelt es sich genaugenommen um ein Verfahren, die Jacobi-Matrix bei einem bestimmten Typ von neuronalen Netzen schneller zu berechnen. Dazu wird der Fehler durch die Netzstruktur „zurückgeschickt“, wodurch bereits berechnete Teilergebnisse nochmals genutzt werden können. Dies wird später noch genauer erläutert (siehe Seite 25). Gradientenabstieg hingegen nutzt die Jacobi-Matrix um eine Funktion bezüglich eines Parametervektors zu minimieren.

Die Idee ist, dem negativen Gradienten der Funktion zu folgen, bis ein Minimum erreicht ist. Offensichtlich ist dieses Verfahren anfällig für lokale Minima , weshalb es einige Modifikationen

[4] In der Regel wird dabei die quadratische Fehlerfunktion $\mathrm{E}(\vec{y}_{\text{Modell}}, \vec{y}_{\text{Prozeß}}) = \|\vec{y}_{\text{Modell}} - \vec{y}_{\text{Prozeß}}\|^2$ verwendet.

gibt, dies zu vermeiden. Die wohl bekannteste dürfte „Simulated Annealing“ [80] sein. Die Iterationsformel zur Minimierung einer beliebigen Funktion f bezüglich eines Parametervektors mit Hilfe des Gradientabstiegs lautet:

$$\vec{p}_{i+1} = \vec{p}_i - \eta \frac{\partial \mathrm{f}(\vec{p})}{\partial \vec{p}}(\vec{p}_i) \tag{3.4}$$

Der Parameter η gibt dabei die Schrittweite des Verfahrens an, und muß problemabhängig[5] gewählt werden. Überträgt man das Verfahren auf die durch Formel (3.3) gegebene Sichtweise, so erkennt man zwei Dinge:

1. Die Jacobi-Matrix ist nun nicht nur vom Parametervektor $\vec{p}$, sondern auch von den Beispieldaten $\vec{u}$, $\vec{y}$ und eventuell vom inneren Zustand $\vec{x}$ abhängig.

2. Nachdem die Messungen unter Umständen verrauscht sind, wäre es möglich, daß einzelne Gradienten stark verfälscht sind. Somit wäre die Bildung eines mittleren Gradienten sinnvoll. Man unterscheidet deshalb im Gradientenabstieg zwischen Einzelschritt und Batch-Verarbeitung . Die Batch-Verarbeitung mittelt dabei den Gradienten über eine größere Population von Meßpunkten, wohingegen das Einzelschrittverfahren immer nur einen Meßpunkt betrachtet.

Setzt man Gleichung (3.3) in Gleichung (3.4) ein, so ergibt sich folgendes Approximationsverfahren:

$$\vec{p}_{i+1} = \vec{p}_i - \eta \left(\frac{\partial \mathrm{E}(\vec{y}, \vec{y}_{\text{Prozeß}})}{\partial \vec{y}}(\vec{y}_i) \cdot \frac{\partial \mathrm{f}_{\vec{p}_i}(\vec{u}_i)}{\partial \vec{p}} + \frac{\partial \mathrm{S}(\vec{p})}{\partial \vec{p}}(\vec{p}_i) \right) \tag{3.5}$$

Da der Betrag des Gradienten im Minimum gegen Null strebt, realisiert diese Formel automatisch eine einfache Schrittweitensteuerung . Der Nachteil ist eine langsame Konvergenz in flachen Minima, da der Gradient früh sehr klein wird und man nur langsam das wirkliche Minimum findet. Die folgende Variante stellt insofern eine sinnvolle Erweiterung dar, als sie dieses Problem lösen will.

Eine Variante des Gradientenabstiegsverfahren: RProp

Die eigentliche Information die man aus dem Gradienten beziehen kann, ist die Richtung in der ein Abstieg erfolgen soll. Man folgt also der entgegengesetzten Richtung des Gradienten. Das RProp-Verfahren von Riedmiller und Braun [79] modifiziert nun die einfache Schrittweitensteuerung durch eine explizite Logik, die in Abhängigkeit von der *Änderung* des Vorzeichens der Fehlerfunktion einen Schrittweitenvektor $\vec{\eta}$ für jeden Parameter einzeln einstellt.

Die Änderung der Schrittweite wird durch die Parameter $0 < \eta_- < 1 < \eta_+$ gesteuert, die multiplikativ auf die Schrittweite wirken. Zusätzlich existieren für die Schrittweite eine untere Schranke $\eta_{\min}$ und eine obere Schranke $\eta_{\max}$. Die Formel (3.4) ändert sich dann zu:

$$\vec{p}_{i+1} = \vec{p}_i - \vec{\eta}_i \cdot \operatorname{sgn}\left(\frac{\partial \mathrm{f}(\vec{p})}{\partial \vec{p}}(\vec{p}_i) \right) \tag{3.6}$$

[5] Im linearen Fall kann z.B. für jeden Schritt η optimal gewählt werden, so daß ein bezüglich dieses Schritts minimales Ergebnis erzielt wird.

Man beachte, daß sowohl die Multiplikation als auch die Vorzeichenfunktion sgn komponentenweise ausgeführt werden müssen. Um den neuen Schrittweitenvektor $\vec{\eta}_{i+1}$ zu erhalten wird nun für jede seiner Komponenten die zugehörige Komponente des Gradienten auf einen Vorzeichenwechsel hin geprüft. Fand kein bzw. ein Vorzeichenwechsel statt, so ergibt sich die neue Komponente aus der alten durch Multiplikation mit η_+ bzw. η_-. Ist der aktuelle oder vorherige Gradient Null, so wird die alte Komponente von $\vec{\eta}_i$ unverändert übernommen. Abschließend wird noch auf die obere bzw. untere Grenze geprüft und gegebenenfalls dort abgeschnitten.

Das RProp-Verfahren erzielt in der Regel bessere Ergebnisse als der Gradientenabstieg. Dennoch soll nicht verschwiegen werden, daß in speziellen Fällen – wenn eine Optimierung nur durch eine spezielle Linearkombination mehrerer Parameter möglich ist – keine Verbesserung oder sogar eine Verschlechterung möglich ist.

Konjugierte Gradienten

Die Herleitung der Minimierungsmethode der Konjugierten Gradienten ist sehr komplex und würde den Rahmen dieser Übersicht sprengen. Da es sich jedoch um eine weit verbreitete und anerkannt gute Methode handelt, soll hier kurz die Basisidee des Algorithmus erläutert werden.

Anhand eines linearen Beispiels kann gezeigt werden, daß beim Gradientenabstieg häufige Richtungswechsel erfolgen. Um dies zu umgehen wird mit Hilfe des Gradienten eine neue Schrittrichtung und Schrittweite berechnet, so daß aufeinanderfolgende Schritte orthogonal bezüglich der Fehlerfläche sind. Im linearen n-dimensionalen Fall kann so das Minimum mit n Schritten gefunden werden. Die Methode zeichnet sich durch eine in der Regel schnellere Konvergenz aus und ist auch auf nichtlineare Probleme übertragbar. Eine hervorragende Einführung und entsprechende Algorithmen können in [92] gefunden werden.

Andere Techniken

Die bisher vorgestellten Methoden zeichnen sich durch Ihre Fähigkeit aus, auch mit hochdimensionalen Parametervektoren umzugehen und sind somit prädestiniert, im Umfeld der neuronalen Netze eingesetzt zu werden. Darüber hinaus existieren noch eine ganze Reihe von anderen Optimierungsverfahren, die im folgenden kurz erwähnt werden sollen:

Verfahren für geringe Parameterzahl. Hier findet man unter anderem verschiedenste Vertreter der Gauß-Newton- [5] und Levenberg-Marquardt- [65] Algorithmen . Letztere Klasse von Verfahren hat sich mittlerweile als Standard für Optimierungsprobleme mit niedriger Parameterzahl herauskristallisiert. Eine Realisierung findet sich in [77].

Verfahren für mittlere Parameterzahl. Hierfür bieten sich Quasi-Newton-Algorithmen an. Zu erwähnen sind die Verfahren von Broyden-Fletcher-Goldfarb-Shanno (BFGS) und Davidson-Fletcher-Powell (DFP). Beide sind sehr ähnlich und unterscheiden sich im wesentlichen nur in der Parametrierung. Es hat sich jedoch gezeigt, daß das BFGS Verfahren in der Praxis leicht überlegen ist. Eine theoretische Einführung und Diskussion beider Verfahren findet sich in [93]. Eine Realisierung des BFGS-Verfahrens ist in [77] dargestellt.

3.3 Neuronale Netze

Die Modellierungsmethodik der Neuronalen Netze ist durch biologische Vorgänge motiviert. Ergebnisse der Hirnforschung sowie allgemeine Überlegungen zur Repräsentation unserer Umwelt im Gehirn haben zu einer Vielfalt von Modellen geführt.

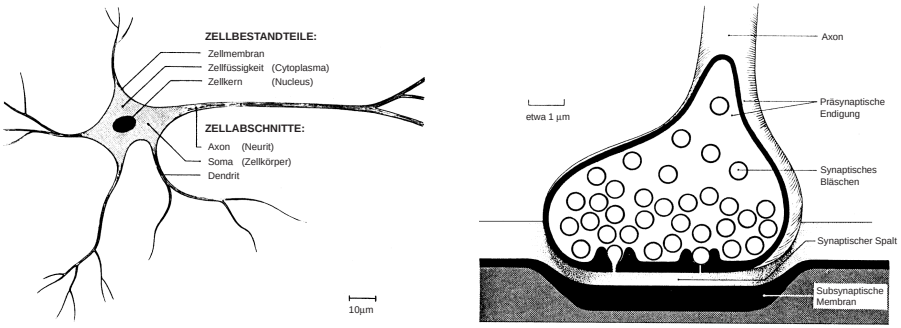


Abbildung 3.2: Ein biologisches Neuron mit Synapse nach [90]. Die einzelnen Neurone sind miteinander über Dendritenbäume verschaltet. Die Anschaltung eines Dendriten an das Axon eines anderen Neurons erfolgt über Synapsen. Dort kann über verschiedenste chemische Stoffe die Reaktion auf ein eingehendes Signals moduliert werden.

Der prinzipielle Aufbau eines *künstlichen* neuronalen Netzes folgt im wesentlichen dem natürlichen Vorbild. Es besteht aus einer Menge von – in der Regel in Schichten angeordneten – Neuronen, die jeweils gegenseitig miteinander verschaltet sind. Die Wirkungsweise einer natürlichen Synapse wird in der Verschaltung dabei normalerweise nur grob durch einen Gewichtswert angenähert, der die Stärke der Verschaltung angibt. Auch die Wirkungsweise des Neurons selbst ist extrem vereinfacht. So wird bei den meisten hier genannten Modellen die mittlere Feuerfrequenz durch einen einzigen Wert angegeben. Der in Abbildung 3.3 gezeigte typische Aktionspotentialverlauf wird ebenfalls nur angenähert. So verwenden gängige Netzwerkmodelle Sprungfunktionen bzw. deren stetige Varianten, um das Depolariationsverhalten nachzubilden.

Im folgenden wird aus dieser Vielfalt von Architekturen ein kleiner Ausschnitt präsentiert, der sich für die zuvor genannten Ziele der Modellbildung besonders eignet. Im Speziellen wird nicht auf neuere Modelle – sogenannte Single-Spike Netze – eingegangen, die auch einzelne Spikes simulieren und zum Beispiel in [97, 21] zu finden sind.

Die vorgestellten Modelle sind bei genauer Betrachtung sehr nahe verwandt zu den aus der Numerik und Statistik bekannten Modellen zur Funktionsapproximation. So kann etwa die Gewichtematrix identisch zum zuvor eingeführten Parametervektor $\vec{p}$ behandelt werden. Somit stehen alle genannten Lernverfahren zur Verfügung, wenn die Jacobi-Matrix der Netzausgabe bezüglich der Gewichte bekannt ist.

3.3.1 Einfache vorwärtsgerichtete Netze

Diese Klasse von neuronalen Netzen eignet sich als Funktionsapproximator statischer Funktionen. Basierend auf den Arbeiten von Stone-Weierstrass und Kolmogorov existieren für die

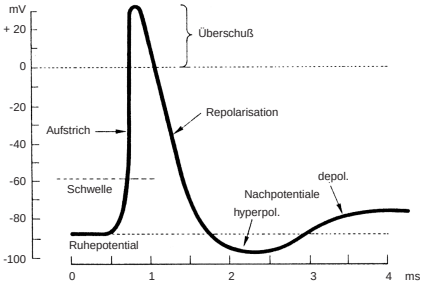


Abbildung 3.3: Der Verlauf eines Aktionspotentials nach [90]. Der Aufstrich, die sogenannte Depolarisation findet statt sobald das Membranpotential einen festen Schwellwert übersteigt. Der gezeigte Verlauf ist dann prinzipiell immer gleich, kann jedoch in seiner speziellen Ausprägung von weiteren Faktoren abhängen.

meisten Architekturen Beweise, die ihre Fähigkeit zur „allgemeinen Funktionsapproximation“ zeigen. So wird dies zum Beispiel in [41] für Multi-Layer-Feedforward-Netze bewiesen. Die Netzgraphen sind dabei zyklenfrei, so daß sie vorwärtsgerichtet abgearbeitet werden können. In der Regel werden die Neuronen schichtweise angeordnet und von Schicht zu Schicht vollständig vernetzt.

Stellvertretend für diesen Typ neuronaler Netze sollen die zwei mit am häufigsten genutzten Varianten, nämlich das Multi-Layer-Perzeptron und die „Radial-Basis-Function“-Netze kurz vorgestellt werden.

Multi-Layer-Perzeptron

Dieses Modell, kurz MLP genannt, wurde erstmals ausführlich von Rumelhart und McClelland [84] besprochen. Die Weiterentwicklung des von Rosenblatt [81, 82] vorgestellten Perzeptrons verfügt über einen schichtenweisen Aufbau. Jede der N Schichten ist mit der nachfolgenden durch die Gewichtematrix $\mathbf{W}_i$ vollvernetzt. Jede Schicht verfügt über einen sogenannten Bias-Vektor $\vec{b}$, der unabhängig von der vorherigen Schicht seinen Wert direkt einbringt. Abbildung 3.4 zeigt den schichtenweisen Aufbau eines Multi-Layer-Perzeptrons.

Die Aktivierungsfunktion der einzelnen Neurone ist in der Regel sigmoid und folgendermaßen definiert:

$$\mathrm{s} : x \mapsto \frac{1}{1+e^{-x}} \quad \text{mit} \quad \frac{\partial \mathrm{s}}{\partial x}(x) = (1-\mathrm{s}(x))\mathrm{s}(x) \tag{3.7}$$

Die Funktion $\mathrm{s}(\vec{x})$ wird im weiteren synonym zu einer komponentenweisen Anwendung von $\mathrm{s}(x)$ auf $\vec{x}$ verwendet. Setzt man eine lineare Ein- und Ausgabeschicht voraus, so ergibt sich folgende rekursive Berechnungsvorschrift für die Abbildungsfunktion $\mathrm{f} : \vec{u} \mapsto \mathrm{y}_N(\vec{u})$:

$$\mathrm{y}_i : \vec{u} \mapsto \begin{cases} \vec{u} & i = 0 \\ \vec{b_i} + \mathbf{W}_i \mathrm{y}_{i-1}(\vec{u}) & i = N \\ \mathrm{s}(\vec{b_i} + \mathbf{W}_i \mathrm{y}_{i-1}(\vec{u})) & \text{sonst} \end{cases} \tag{3.8}$$

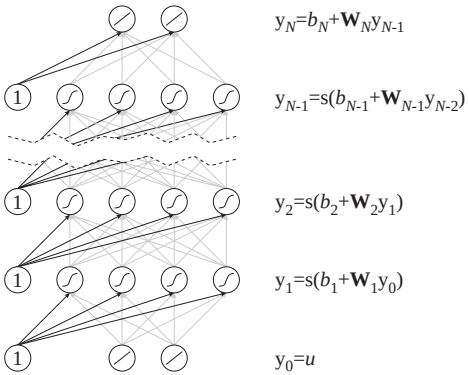


Abbildung 3.4: Struktur eines Multi-Layer-Perzeptrons . Die Ein- und Ausgabeschicht enthalten lineare Aktivierungsfunktionen. Alle übrigen Aktivierungsfunktionen sind sigmoid. Der Bias Vektor $\vec{b}$ wird jeweils über ein konstantes Neuron der Vorgängerschicht mit dem Wert Eins eingeschleust und ist schwarz eingezeichnet. Die Koppelungen der Gewichtematrix $\mathbf{W}$ hingegen sind dunkelgrau markiert.

Backpropagation Die Berechnung der zum Lernen notwendigen Jacobi-Matrix erfolgt ebenfalls rekursiv, jedoch in umgekehrter Richtung. Dabei wird der am Ausgang des Netzes auftretende Approximationsfehler rückwärts durch die Netzstruktur geschickt und dabei gewichtet auf die vorgeschalteten Neurone verteilt. Es handelt sich dabei um den Backpropagation-Algorithmus aus [83]. Die notwendigen Ableitungen zum Belegen der Jacobi-Matrix in Matrixform erhält man nach [98] aus folgenden Rekursionsformeln:

$$\begin{aligned}
\mathbf{Y}'_{i-1} &= \mathbf{Y}'_i \cdot \mathbf{S}'_i \cdot \mathbf{W}_i \\
\frac{\partial y_N}{\partial (\mathbf{W}_i)_{jk}} &= \mathbf{Y}'_i \cdot \mathbf{S}'_i \cdot \begin{pmatrix} (y_{i-1})_k \\ \vdots \\ (y_{i-1})_k \end{pmatrix} \\
\frac{\partial y_N}{\partial \vec{b}_i} &= \mathbf{Y}'_i \cdot \mathbf{S}'_i
\end{aligned} \tag{3.9}$$

Dabei enthält die Diagonalmatrix $\mathbf{S}'_i$ die Ableitungen der Aktivierungsfunktionen der $i$ten Schicht. Im Falle der sigmoiden Aktivierungsfunktion $s(x)$ kann diese ohne Neuberechnung aus (3.7) erhalten werden. Die Matrix $\mathbf{Y}'_i$ enthält die Ableitungen der Nten Schicht (=Ausgabeschicht) nach der $i$ten Schicht. Zu Beginn der Rekursion ist $\mathbf{Y}'_N$ mit einer Einheitsmatrix zu initialisieren.

Problematisch beim Belernen dieser Netzarchitektur ist die notwendigerweise hohe Zahl von Trainingsbeispielen und eine lange Trainingsphase. Dies kann durch unzureichend normierte Datenvektoren noch verstärkt werden, da der Gradient der Sigmoidfunktion (3.7) in der Sättigung, bei betragsmäßig großem x gegen Null geht . Der Backpropagation-Algorithmus kann in Kombination mit Gradientenabstieg bei einer Neuronenzahl von n mit einer Komplexität von $O(n^2)$ implementiert werden.

QuickProp Dieses von Fahlman [26] vorgestellte Verfahren hat eine Konvergenzbeschleunigung des Lernens im Vergleich zum Backpropagation-Algorithmus in Kombination mit Gradientenabstieg zum Ziel. Dabei kommt eine Heuristik zum Einsatz, die ausnutzt, daß in flachen Minima die Krümmung, d.h. die zweite Ableitung, ebenfalls klein ist. Skaliert man den Gradienten mit der inversen zweiten Ableitung, so erhält man auch bei kleinem Gradienten in flachen Minima eine größere Schrittweite. Dies wird in diesem Fall durch die Annahme einer lokal quadratischen Fehlerfunktion und anschließendem direktem Sprung zum (geschätzten) Minimum realisiert.

„Radial-Basis-Function"-Netze (RBF)

Dieses erstmals von Powell [76] und später etwa zeitgleich von Poggio und Giorosi [75] und Moody und Darken [64] nochmals vorgestellte Netz besteht aus nur einer versteckten Schicht. Diese enthält Gaußsche Glocken als Aktivierungsfunktionen. Diese sind mit je einem Mittelwert $\vec{\mu}_i$ und einer gemeinsamen[6] Streuung σ parametriert. Abbildung 3.5 zeigt den Aufbau eines solchen Netzes.

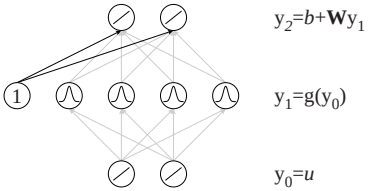


Abbildung 3.5: Die Struktur eines „Radial-Basis-Function"-Netzes benötigt aufgrund der Gaußschen Aktivierungsfunktion $g(x)$ keinen Bias Vektor in der ersten Schicht. In der linearen Ausgabeschicht wird der Bias Vektor $\vec{b}$ wie zuvor über ein konstantes Neuron der Vorgängerschicht mit dem Wert Eins eingeschleust.

Definiert man die Gaußsche Glockenfunktion wie üblich als $g : \vec{x} \mapsto e^{-\frac{\|\vec{x}-\vec{\mu}_i\|^2}{\sigma^2}}$, so erhält man die Abbildungsfunktion eines „Radial-Basis-Function"-Netzes mit linearer Ausgabeschicht als:

$$f_{\text{linear}} : \vec{u} \mapsto \vec{b} + \mathbf{W} \begin{pmatrix} g_1(\vec{u}) \\ \vdots \\ g_m(\vec{u}) \end{pmatrix} \tag{3.10}$$

Als Variante bietet sich eine normalisierte Version an. Die Aktivierungen der versteckten Schicht werden durch die Gesamtaktivierung dieser geteilt, und man erhält folgende modifizierte Darstellung:

$$f_{\text{normalisiert}} : \vec{u} \mapsto \vec{b} + \mathbf{W} \begin{pmatrix} g_1(\vec{u}) \\ \vdots \\ g_m(\vec{u}) \end{pmatrix} \frac{1}{\sum_{i=1}^{m} g_i(\vec{u})} \tag{3.11}$$

[6] Es existieren auch Varianten mit getrennten Streuungen für je eine Gaußsche Glocke. Darüberhinaus läßt sich die Streuung auch durch eine Kovarianzmatrix ersetzen, was zu ellipsoiden Formen führt. In der Praxis hat sich jedoch gezeigt, daß die dadurch gewonnene Flexibilität die zusätzliche Parameterkomplexität nicht aufwiegen kann.

Diese Netzwerktopologie hat sich als sehr gute Architektur für Funktionsapproximationen erwiesen. Aufgrund der flachen Hierarchie ist besonders das Training in annehmbarer Zeit durchführbar. Der lokale Charakter der Approximation führt zu Vorteilen beim Erlernen von Daten unterschiedlicher Dynamik. In der Regel werden die Zentren der ersten Schicht aus den Trainingsdaten mit Hilfe von Clusterverfahren generiert. Eine Übersicht verschiedener Variationen von RBF-Netzen findet sich in [14].

3.3.2 Rekurrente Netze

Die bisher betrachteten Architekturen waren rein vorwärtsgerichtete Architekturen, die – sofern sie mehrschichtig sind – in einer bestimmten Reihenfolge abgearbeitet werden müssen, bis das Ergebnis zu Verfügung steht. Dies erfolgt erst einmal ohne Berücksichtigung eines *externen* zeitlichen Verlaufs oder Taktes. Insbesondere sind diese Systeme zustandsfrei[7] und definieren sich rein über ihre Parametrierung und den Eingabevektor.

Nun soll eine Klasse von Netzen eingeführt werden, die über einen internen Zustand – nämlich die aktuelle Aktivierung der einzelnen Neuronen – verfügen, der sich auch auf zukünftige Berechnungen auswirkt. Dies erreicht man, indem sogenannte rekurrente Verbindungen zugelassen werden, womit man ebenfalls die Schichtenstruktur fallen läßt. Somit existiert nur noch eine Gewichtematrix, die bestimmt, wie Neuron i mit Neuron j verschaltet ist. Desweiteren wird ein externer Takt der Frequenz $\frac{1}{\tau}$ eingeführt, der jeweils die Zeitpunkte einer Neuberechnung der Neuronenaktivierungen vorgibt. Implizit versieht man dadurch die einzelnen Verbindungen zwischen den Neuronen mit einer Signallaufzeit von τ. Mathematisch gesehen erhält man so eine Euler-Simulation eines nichtlinearen Differentialgleichungssystems mit der Schrittweite τ.

Umgekehrt heißt das aber, daß mit Kenntnis der Regelstrecke und damit des zugehörigen Differentialgleichungssystems ein passendes rekurrentes Netz konstruiert werden kann. Sind dann noch Parameteranpassungen notwendig, so könnten diese mit Lernverfahren für rekurrente Netze durchgeführt werden. Dies war bisher nicht möglich, da die gängigen Verfahren *alle* Parameter adaptieren. Dieses Problem wurde mit Hilfe der sogenannten „vorstrukturierten rekurrenten Netze“ [8, 9, 10] behoben. Hier ist es möglich, einzelne Netzgewichte zu fixieren und vom Lernen auszunehmen. Abbildung 3.6 zeigt als Beispiel eines solchen Netzes einen „neuronalen“ PI-Regler.

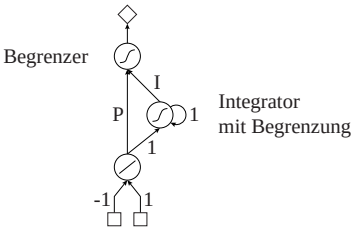


Abbildung 3.6: Das abgebildete rekurrente Netz realisiert einen PI-Regler mit Begrenzung des Integratorwertes und der Ausgabe durch eine Sigmoidfunktion.

[7] Man kann zwar auch hier – wie im folgeden erläutert – die Aktivierung der einzelnen Neuronen als internen Zustand betrachten, diese sind dann jedoch vollständig durch die *aktuelle* Eingabe bestimmt und haben keine Auswirkung auf zukünftige Berechnungen.

Elman- und Jordan-Netze

Eine Vorstufe zu den voll rekurrenten Netzen stellen die von Jordan [46] und Elman [25] untersuchten Architekturen dar. Bei beiden wird durch rekurrente Verbindungen Information über den zeitlichen Verlauf der Netzaktivierungen und damit über den inneren Zustand des Systems gewonnen. Der Vorteil der verwendeten Netzstrukturen ist, daß keine speziellen Lernverfahren verwendet werden müssen, wie es etwa bei den zuvor beschriebenen rekurrenten Netzen der Fall ist.

Beim Jordan- bzw. Elman-Netz handelt es sich um ein erweitertes Multi-Layer-Perzeptron mit einer versteckten Schicht. Zusätzlich zur normalen Eingabe x erhält das Netz noch sogenannte Kontextinformationen. Das in Abbildung 3.7 gezeigte Jordan-Netz realisiert dies durch eine um τ verzögerte Rückführung der Ausgabeschicht in die Kontextschicht. Diese enthält dann Informationen über den aktuellen Zustand des Netzes, die durch Kumulation der letzten Ausgabe mit der letzten Aktivierung der Kontextschicht entstanden ist. Im Gegensatz dazu wird beim Elman-Netz die versteckte Schicht in die Kontextschicht kopiert.

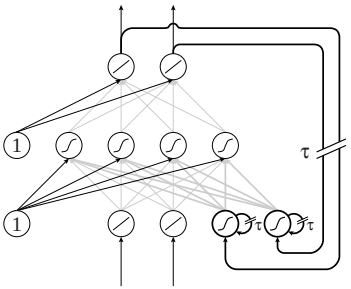


Abbildung 3.7: Der Aufbau eines Jordan-Netzes. Die Erweiterungen gegenüber einem Multi-Layer-Perzeptron sind **fett** markiert.

Zum Belernen der Netze ignoriert man die rekurrenten Verbindungen und erhält ein normales Multi-Layer-Perzeptron, welches mit den zugehörigen Lernverfahren trainiert werden kann. Der Nachteil dieser Vorgehensweise ist, daß über die rekurrenten Verbindungen eigentlich ebenfalls ein Fehlerrückfluß stattfinden würde, der nun unterbunden ist. Dadurch ist dieser Netztyp für komplexere Dynamiken nur sehr eingeschränkt nutzbar.

Lernverfahren

Es werden nun die zwei bekanntesten Lernverfahren für rekurrente Netze kurz vorgestellt. Zuvor soll aber noch darauf hingewiesen werden, daß die vorgestellten Trainingsverfahren für rekurrente Netze auf verrauschte Lerndaten und Ausreißer sehr empfindlich reagieren. Dies liegt an der rückgekoppelten Verarbeitung, welche kleine Abweichungen häufig so potenziert, daß völlig unerwartete Ergebnisse produziert werden. Auch sogenannte Langzeitabhängigkeiten sind mit normalen rekurrenten Netzen nicht sinnvoll lernbar, da der Fehlergradient mit zunehmender Sequenzlänge verschwindet [40].

Backpropagation Through Time (BPTT) Bei diesem Ansatz wird das Lernen auf das Lernen eines vorwärtsgerichteten Netzes zurückgeführt [111]. Dazu kann durch eine zeitliche

Entfaltung das rekurrente Netz in diese Form überführt werden. Da eine unendliche Entfaltung aufgrund des unendlichen Speicherbedarfes nicht möglich ist, muß nach einer festen Zahl von Zeitschritten abgebrochen werden. Wie bei Jordan-Netzen wird dadurch auch hier der Fehlerrückfluß unterbunden. Ist die zu lernende Sequenz kürzer als diese maximale Zahl der Zeitschritte, oder die Ordnung des zu approximierenden Systems niedrig genug, so kann korrekt gelernt werden.

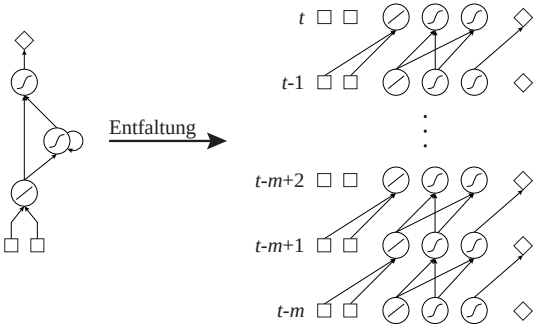


Abbildung 3.8: Der PI-Regler aus Abbildung 3.6 wird zeitlich entfaltet. Man erhält ein mehrschichtiges Netz, das z.B. mit Gradientenabstieg und Backpropagation belernt werden kann.

Abbildung 3.8 zeigt die Entfaltung des zuvor gezeigten PI-Reglers. Man sieht sofort, daß diese Technik für längere Zeitsequenzen zu einem stark erhöhten Speicherbedarf führt. Auch die Rechenzeitkomplexität erhöht sich dementsprechend. Setzt man n für die Zahl der Neuronen und m für die Zahl der Entfaltungsschritte, so ergibt sich eine Speicherkomplexität von $O(mn + n^2)$ und eine Berechnungskomplexität von $O(n^2 m)$.

Real Time Recurrent Learning (RTRL) Im Gegensatz zum vorherigen Algorithmus eignet sich dieses Verfahren für beliebige Sequenzlängen, und arbeitet mit einem korrekten Fehlergradienten zum Lernen. Die Berechnungsvorschrift eines rekurrenten Netzes in Matrixform ist gegeben durch:

$$\vec{y}(t) = \mathrm{s}(\mathbf{W}\vec{y}(t-1)) \tag{3.12}$$

Dabei steht $\vec{y}(t)$ für die Aktivierungen des Netzes zum Zeitpunkt t. Die in der Regel dünn besetzte $n \times n$ Gewichtematrix $\mathbf{W}$ bestimmt die Verschaltung des Netzes. Die Aktivierungsfunktion $\mathrm{s} : \mathbb{R}^n \to \mathbb{R}^n$ kann für jedes Neuron (= pro Dimension) separat gewählt werden. Obige Darstellung ist identisch zu den in [110, 111] vorgestellten Versionen. Durch die gewählte Matrixschreibweise vereinfacht sich die Darstellung wesentlich und ist nach [8] auch für Aktivierungsfunktionen mit mehreren Eingaben gültig. Ausgehend vom obigen Verarbeitungsschritt erhält man nun die zur Adaption der Gewichte notwendige $n \times n^2$ Jacobi-Matrix $\mathbf{J}(t)$ durch folgende Iterationsgleichung:

$$\mathbf{J}(t) = \mathbf{S}'(t)(\mathbf{W} \cdot \mathbf{J}(t-1) + \mathbf{Y}(t-1)) \tag{3.13}$$

Die Iteration startet mit $\mathbf{J}(t=0)$ als Nullmatrix. Wie in Formel (3.9) enthält $\mathbf{S}'(t)$ wiederum die Ableitungen der Aktivierungsfunktionen. Die Matrix $\mathbf{Y}(t-1)$ enthält die Aktivierungen zum Zeitpunkt $t-1$. Die $i$te Zeile enthält dabei an n Positionen die Aktivierung $y_i(t-1)$ des $i$ten Neurons und ansonsten Null. Die einzelnen besetzten Positionen hängen davon ab, wo die dem entsprechenden Neuron vorgeschalteten – und damit in Bezug auf die Ableitung relevanten – Gewichte im Parametervektor bzw. in der Jacobi Matrix abgelegt sind.

Der Vorteil dieses Verfahren ist, daß der Gradient für beliebige Sequenzlängen korrekt berechnet wird. Darüberhinaus muß keine Vergangenheitsinformation verwaltet werden. Die Speicherkomplexität beträgt jedoch immer noch $O(n^3)$. Die Zeitkomplexität ist mit $O(n^4)$ ebenfalls sehr hoch. Es existieren aber Kombinationsverfahren aus BPTT und RTRL, die eine Zeitkomplexität von $O(n^3)$ aufweisen und dennoch mit korrektem Fehlergradienten arbeiten [89].

Teacher Forcing Diese, speziell bei rekurrenten Netzen, sinnvolle Erweiterung Lernen zu beschleunigen, ist in [110] dargestellt. Im wesentlichen wird für bekannte, weil vorgegebene, Aktivierungsverläufe einzelner Neurone dieser anstelle der wirklichen Aktivierung zur Berechnung der Jacobi-Matrix verwendet. Dieses Vorgehen vermeidet Effekte wie Drift oder Potenzierung von Rauschen, die ansonsten im sogenannten „Free-Run“ entstehen können.

3.3.3 Andere Netzarchitekturen

Neben den bisher erwähnten Netzarchitekturen existiert eine Unzahl von weiteren Spezialarchitekturen, die jeweils auf bestimmte Probleme zugeschnitten sind. Im folgenden sollen zuerst einige Netze vorgestellt werden, die sich zur Identifikation von Dynamiken bekannter Ordnung bzw. von Totzeiten nutzen lassen. Abschließend soll noch ein Netz vorgestellt werden, das entwickelt wurde, um minimale Modelle zu finden.

Time Delay Neural Network (TDNN)

Im dynamischen Umfeld findet man häufig eine Architektur vor, die nicht nur den letzten Eingabevektor präsentiert, sondern auch vorherige Eingaben [56]. Das können entweder die k letzten oder beliebig verzögerte Kombinationen sein. Effektiv wird dazu die bisherige Modellfunktion von $\mathrm{f} : \vec{u}_i \mapsto \vec{y}_i$ durch eine Modifikation des Eingabevektors zu $\mathrm{f} : \{\vec{u}_{\pi_1}, \vec{u}_{\pi_2}, \ldots \vec{u}_{\pi_k}\} \mapsto \vec{y}_i$ geändert. Dabei gibt $i \geq \pi_1 > \pi_2 > \ldots \pi_k$ eine Selektion von k Werten der letzten verfügbaren Eingaben an.

Der Vorteil dieser Vorgehensweise ist die Möglichkeit, einfache vorwärtsgerichtete Netze zu verwenden. Besitzt man ausreichend Vorwissen, so kann sogar die Wahl der π_i gezielt erfolgen. Nachteilig wirkt sich eine unbekannte Ordnung des zu approximierenden Systems aus, da ein unkontrolliertes Aufblähen des Eingabevektors die Trainingszeiten erhöht.

Lokal rekurrente Systeme

Dieser Typ von Netzen kann ebenfalls von einfach vorwärtsgerichteten Netzen abgeleitet werden. Die Fähigkeit dynamische Systeme zu approximieren wird hierbei durch lokal rekurrente Verbindungen der einzelnen Neuronen erreicht. Meist wird dadurch eine lineare Dynamik 2. Ordnung – auch unter der Bezeichnung (ARMA) bekannt – realisiert. Einerseits erhält man dadurch Modelle mit internen Zuständen, andererseits läßt sich das auf lokal rekurrente Verbindungen beschränkte Modell leichter adaptieren, als sein vollvernetztes Pendant.

Als Beispiele für Netze dieser Art seien hier die dynamischen Varianten des Multi-Layer-Perzeptrons (DMLP) und des „Radial-Basis-Function"-Netzes (DRBF) genannt, deren Architektur und Adaptionsverfahren in [3, 2] beschrieben ist. Abbildung 3.9 zeigt die Verschaltung eines solchen dynamischen Neurons.

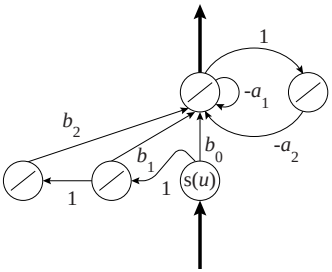


Abbildung 3.9: Dynamisches Neuron mit ARMA-Filter zweiter Ordnung. Die Verschaltung ist als neuronales Netz ausgelegt, befindet sich allerdings in der Regel innerhalb eines Neurons. Die Adaption der ARMA Parameter erfolgt lokal, so daß alte Gradienten ebenfalls nur lokal im Neuron gespeichert werden müssen. Die Aktivierungsfunktion s(u) kann beliebig gewählt werden, da der ARMA-Filter nachgeschaltet ist.

Adaptive Time Delay Neural Network (ATNN)

Es bietet sich an, statt der fixen Zeitverzögerungen der Time Delay Neuronalen Netze, variable und adaptierbare Zeitverzögerungen wie in Abbildung 3.10 zu nutzen. Von Lin, Dayhoff, und Ligomedes [58] wurde eine entsprechende Architektur vorgestellt und in [59] entsprechende Ergebnisse hierzu veröffentlicht. Leider ist die präsentierte Herleitung des Lernalgorithmus stellenweise lückenhaft und teilweise nicht nachvollziehbar. Der resultierende Algorithmus deckt sich jedoch mit der sauberen Herleitung aus Day und Davenport [19].

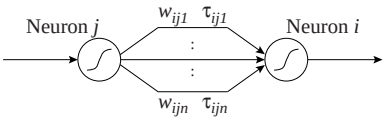


Abbildung 3.10: Im Gegensatz zum normalen Multi-Layer-Perzeptron werden zwei Neuronen im ATNN-Netz nun mit n Gewichten und n Zeitverzögerungen miteinander verbunden.

Zur Vereinfachung wird im folgenden die übliche Schichtenstruktur im Multi-Layer-Perzeptron vernachlässigt und ein Netz mit einer einzigen Gewichtsmatrix betrachtet. Statt der bisher einfachen Verbindung zweier miteinander verschalteter Neurone i und j, existieren nun n Verbindungen, die mit jeweils eigenen Gewichten und eigenen variablen Zeitverzögerungen τ_k versehen sind. Die Berechnungsfunktion für die Ausgabe des $i$ten Neurons ändert sich damit zu:

$$\mathrm{y}_i(t) = \mathrm{s}\left(\sum_{j,k} w_{ijk}(t) \cdot \mathrm{y}_j(t - \tau_{ijk}(t))\right)$$

Man beachte die ungewöhnliche Schreibweise der Parameter w und τ: Sie erhalten aus formalen Gründen eine Zeitabhängigkeit, so daß Ableitungen nach t möglich sind. Aufgrund der variablen Verzögerungszeiten können nun auch Effekte wie Totzeiten modelliert werden. Die Herleitung der zum Lernen notwendigen Jacobi-Matrix ist in [104] beschrieben und liefert für die einzelnen Einträge folgendes Ergebnis:

$$\begin{aligned} \frac{\partial \mathrm{y}_a(t)}{\partial \tau_{ijk}(t)} &= -\delta_{ai}(t) \cdot w_{ijk}(t) \cdot \mathrm{y}'_j(t - \tau_{ijk}(t)) \\ \frac{\partial \mathrm{y}_a(t)}{\partial w_{ijk}(t)} &= \delta_{ai}(t) \cdot \mathrm{y}_j(t - \tau_{ijk}(t)) \end{aligned} \tag{3.14}$$

Dabei wird der Index a synonym für Ausgabeneurone eingesetzt, und die Notation y' bezeichnet die Ableitung $\frac{\partial \mathrm{y}(t)}{\partial t}$ nach der Zeit[8]. Die Ableitung des $a$ten Ausgabeneurons nach der Eingabe des $i$ten Neurons ist mit $\delta_{ai}(t)$ bezeichnet und entspricht dem zurückpropagierten Fehler. Sie kann über eine ähnliche Rekurrsionsgleichung wie die in Formel (3.9) gezeigte berechnet werden und ersetzt sinngemäß die dort verwendete Größe $\mathbf{Y}'_i \cdot \mathbf{S}'_i$.

Die in Gleichung (3.14) gezeigten Ableitungen sind kausal inkonsistent, da zum Lernen teilweise zukünftige Werte benötigt werden [104]. Aus diesem Grund wird ein Verzögerungswert – auch Aging Wert genannt – eingeführt, der den kausalen Zusammenhang wieder herstellt.

Es hat sich leider gezeigt, daß sich – trotz korrekter Modellierung des Systems – aus den in den Netzen nach dem Lernen befindlichen Verzögerungszeiten keine direkten Rückschlüsse auf die im modellierten System vorhandenen Totzeiten möglich sind. Die Lernzeiten sind vergleichbar zu einem normalen Multi-Layer-Perzeptron, wenn man die erhöhte Parameterzahl berücksichtigt. Das Lernen variabler Totzeiten ist mit diesem Ansatz nicht möglich.

Weitere Ergebnisse zum Einsatz dieses Netzes in der Regelungstechnik können in [105, 106] gefunden werden.

Long Short-Term Memory (LSTM)

Das Problem rekurrenter Netze durch verschwindende Fehlergradienten über mehrere Zeitschritte hinweg wurde bereits angesprochen. Dieses Verhalten ist bedingt durch Gewichte die betragsmäßig kleiner eins sind. Für Gewichte die betragsmäßig größer als eins sind, stellt sich aber instabiles Verhalten beim Lernen ein. Sinnvoll ist es, Information explizit zu speichern, um auch Langzeitabhängigkeiten lernen zu können. Dies ist in LSTM über eine spezialisierte neuronale Speicherzelle realisiert, die in Abbildung 3.11 gezeigt ist.

Sie enthält im wesentlichen einen Integrator, der als Informationsspeicher dient. Um nicht jede Information zu speichern bzw. jederzeit zur Verfügung zu stellen, verfügt die Zelle über multiplikative Synapsen – auch „Gates“ genannt – die den Informationsfluß ein- und ausschalten können. In [40] wird gezeigt, daß mit Hilfe dieser Architektur auch bisher nicht lernbare extrem lange zeitliche Abhängigkeiten gelernt werden können.

[8] Diese Ableitung steht nicht unmittelbar zur Verfügung, und muß durch den Differenzenquotienten genähert werden.

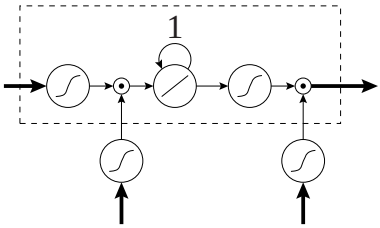


Abbildung 3.11: Schaltbild der in LSTM verwendeten Speicherzelle. Besonderheiten sind einerseits die multiplikativen Synapsen (markiert als Kreis mit Punkt), die verwendet werden, um den Informationsfluß ein- und auszuschalten. Andererseits wird ein Integrator über eine lineare Aktivierungsfunktion und lokale Rekurrenz mit fixiertem Gewicht von 1 realisiert, um den konstanten Fehlerrückfluß und die Informationsspeicherung zu gewährleisten.

Flat Minimum Search (FMS)

Dieses Verfahren stellt eine Alternative zu den im nächsten Abschnitt vorgestellten Algorithmen zur Vermeidung des Bias-Variance-Problems dar. Die Idee ist, minimale Netze mit möglichst unscharfen Gewichten zu generieren, was durch einen speziellen Strafterm in Kombination mit einem Löschverfahren für unnötige Gewichte *bereits zur Laufzeit* erreicht wird.

Die dadurch entstehenden „flachen Minima“ haben diesem Netzwerk seinen Namen gegeben. Eine genaue Analyse dieser Architektur und ausführliche Vergleiche sind in [40] zu finden.

3.3.4 Ansätze zur Vermeidung des Bias-Variance-Problems

Dieses bereits ausführlich in Abschnitt 3.2.1 angesprochene Problem tritt wegen seiner grundsätzlichen Natur auch bei Neuronalen Netzen auf. Neben den allgemein anwendbaren Methoden zur Linderung dieses Problems existieren einige bei Neuronalen Netzen häufig eingesetzte Verfahren, von denen eine Auswahl vorgestellt wird.

Weight-Decay

Diese Verfahren haben Ihre Ursprünge in der Statistik und sind dort unter dem Begriff Regularisierung [103] bekannt. In Gleichung (3.3) wurde der Regularisierungsanteil als $S(\vec{p})$ geschrieben. Nun soll aufgrund der etwas anderen Sichtweise synonym hierfür $S(\vec{w})$ verwendet werden. Der Gewichtevektor $\vec{w}$ enthält dabei alle im Netz vorkommenden Gewichte. Ziel ist es, durch spezielle Terme Kriterien wie etwa Glattheit zu fordern.

Gängige Weight-Decay-Terme sind zum Beispiel der von Hanson und Pratt [36] vorgestellte quadratische Strafterm $S(\vec{w}) = \lambda\|\vec{w}\|^2$, der kleine Gewichte stark bevorzugt und damit einen glatteren Verlauf erzeugt[9].

Von Weigend, Rumelhart, und Huberman [109] wurde $S(\vec{w}) = \lambda \sum_i \frac{w_i^2}{\hat{w}^2+w_i^2}$ als Strafterm vorgeschlagen. Mit Hilfe des Parameters $\hat{w}$ kann zwischen der Forderung nach weniger, aber großen Gewichten ($\equiv \hat{w}$ klein) und vielen, aber kleinen Gewichten ($\equiv \hat{w}$ groß) variiert werden.

[9] Diese Form ist auch als Momentum-Term bekannt und beschränkt durch kleine Gewichte die höheren Ableitungen des Netzes, was wiederum zu einem glatterem Verlauf führt.

Ein Strafterm, der die Varianz der Ausgabe der Zwischenschicht-Neuronen minimiert, wurde von Schittenkopf, Deco, und Brauer [86] vorgestellt. Dadurch wird Rauschen der Eingabedaten eliminiert und Überlernen vermieden. Zusätzlich läßt sich die dort vorgestellte Technik der Varianz-Minimierung auch mit den im folgenden vorgestellten Verfahren nutzen.

Optimal Brain Damage (OBD), Optimal Brain Surgeon (OBS)

Es hat sich gezeigt, daß – neben der Beeinflussung des Fehlergradienten während des Lernens – die Modelle auch nach dem Lernen datenunabhängig modifiziert werden können, so daß es sich positiv auf die Modellgüte auswirkt. Die Idee dabei ist, Gewichte, die sich kaum auf das Modellverhalten auswirken, zu entfernen, was auch mit dem Begriff „Pruning“ bezeichnet wird.

Die beiden folgenden Verfahren berechnen hierzu eine Taylor-Entwicklung der Änderung der Fehlerfunktion $\partial \mathrm{E}$ bei infinitesimal kleiner Änderung der Gewichte $\partial\vec{w}$ mit Termen bis zum 2ten Grad:

$$\partial \mathrm{E} = \underbrace{\left(\frac{\partial \mathrm{E}}{\partial \vec{w}}\right)^T \cdot \partial\vec{w}}_{\to 0 \text{ im lokalen Minimum}} + \frac{1}{2}(\partial\vec{w})^T \cdot \mathbf{H} \cdot \partial\vec{w} + O(\|\partial\vec{w}\|^3) \tag{3.15}$$

Dabei steht $\mathbf{H}$ für die Hesse Matrix der Fehlerfunktion E. Vernachlässigt man Terme höherer Ordnung und befindet sich nahe an einem lokalem Minimum, so gilt nach [17] folgende Approximation bei Modifikation des $i$ten Gewichtes, die zum OBD-Verfahren führt:

$$\partial E \approx \frac{1}{2} h_{i,i} (\partial w_i)^2$$

Eine Verfeinerung dieses Verfahren stellt OBS nach [39] dar. Hierbei wird eine genauere Approximation für $\partial \mathrm{E}$ und ein Kompensationsterm für den Gewichtevektor genutzt, so daß nach dem Löschen eines Gewichtes noch eine Korrektur der dadurch induzierten Fehler durchgeführt werden kann. In der folgenden Gleichung sind beide Terme dargestellt, der Vektor $\vec{e}_i$ ist dabei der $i$te Einheitsvektor:

$$\partial \mathrm{E} \approx \frac{w_i^2}{2 \cdot (\mathbf{H}^{-1})_{i,i}} \qquad \partial\vec{w} \approx -\frac{w_i}{(\mathbf{H}^{-1})_{i,i}} \mathbf{H}^{-1} \vec{e}_i$$

Der eigentliche Löschalgorithmus ist nun bei beiden Verfahren identisch: Finde dasjenige w_i mit dem kleinsten $\partial \mathrm{E}$. Wenn $\partial \mathrm{E}$ kleiner als eine vorgegebene Schranke ist, so lösche w_i. Bei OBS wird anschließend noch die Korrektur der verbleibenden Gewichte durchgeführt.

3.4 Probleme bei der Modellbildung

Nachdem bereits einige Probleme und zugehörige Lösungsansätze, die im Rahmen einer Modellbildung auftreten können angesprochen wurden, wird nun nochmals kurz auf einige spezielle Probleme, die in dieser Arbeit behandelt werden, eingegangen.

3.4.1 Behandlung von Rauschen, Ausreißern und Sprüngen

Zwei wesentliche Probleme sind die Behandlung von Rauschen und Ausreißern. Sowohl Rauschen, als auch Ausreißer sind Signalanteile, die keinen Informationsgehalt tragen.

Ist die Dynamik des Rauschens bekannt, so kann häufig ein spezieller Filter eingesetzt werden, der – soweit möglich – das Nutzsignal extrahiert. In der Regel ist die Dynamik des Rauschens jedoch unbekannt und man nimmt normalverteiltes weißes Rauschen an. Beim Einsatz bestimmter Lernverfahren kann ein impliziter, teilweise auch unerwünschter, Filtereffekt auftreten. Dies trifft zum Beispiel auf Batch-Verfahren zu, die den Gradienten über eine größere Population von Meßpunkten mitteln.

Eine weitere Störung stellen die sogenannten Ausreißer dar. Dabei handelt es sich entweder um einzelne Meßfehler, oder um Störungen wie sie etwa durch periodisch, aber immer nur kurz, auftretende Ereignisse induziert werden können. Typisch wäre etwa die Störung einer Messung durch den Zündfunken des Motors. Auch hier tritt wieder das Problem der Identifikation des Ausreißers auf. Im Besonderen könnte es sich ja um einen Anteil des Nutzsignals, einen wirklichen Sprung in der Dynamik handeln. Damit kommt man zum nächsten Problem: Alle gängigen Verfahren sind auf stetige Funktionen ausgelegt und können Sprünge nur in Grenzfällen darstellen. Dies äußert sich einerseits in ungewolltem Glätten des Sprungs, andererseits steigen die notwendigen Lernzeiten drastisch an und man stößt auf numerische Probleme. Abbildung 3.12 zeigt als Beispiel die Approximation eines Sprunges durch RBF-Netzwerke.

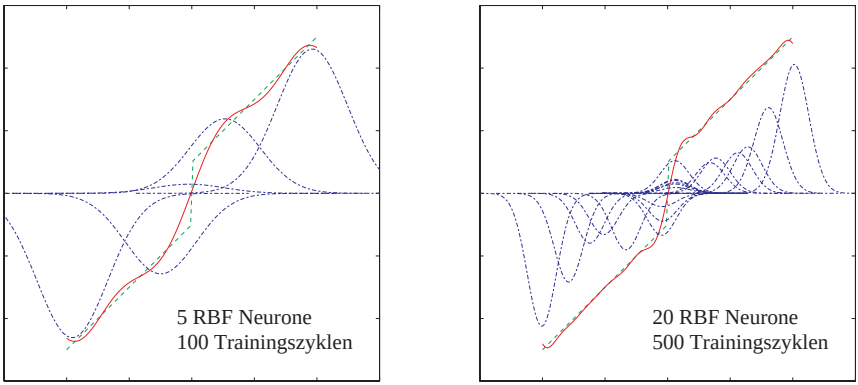


Abbildung 3.12: Die Approximation einer Sprungfunktion erfordert übermäßig viele RBF-Neurone und ist immer ungenau, da implizit geglättet wird. Im gezeigten Beispiel wurden pro Trainingszyklus je 1000 Datensätze präsentiert bis keine Verbesserung mehr eintrat. Die rote Linie zeigt das erstellte Modell, die grün gestrichelte die Originalfunktion. Die erlernten Gauß-Glocken sind blau gepunktet markiert und in der Höhe mit dem entsprechenden Gewichtswert skaliert.

Im nächsten Kapitel wird ein Filter vorgestellt, der in Kombination Rauschen und Ausreißer behandeln kann und dennoch Sprünge zuläßt. Wie gezeigt wird, bringt die Kombination der Behandlung einige Vorteile mit sich. In Kapitel 5 findet sich dann ein Modellbildungsverfahren, das auch mit expliziten Sprüngen bzw. Dynamikwechseln umgehen kann.

3.4.2 Extrapolation bei unbekannten Eingaben

Die im zweiten Teil des nächsten Kapitels kurz angesprochenen Methoden zur Meßplanung greifen ein Problem auf, das in vielfältiger Weise untersucht wird: Soll ein Modell Aussagen über unbekannte Regionen machen, so kann es zu völlig unsinnigen Ergebnissen kommen. Aus diesem Grund versucht man den Eingaberaum möglichst genau zu erfassen und alle relevanten Messungen durchzuführen. Kann das Modell Aussagen über seine eigene Konfidenz machen, so muß nicht zwingend der gesamte Eingaberaum vermessen werden, da in Regionen mit niedriger Konfidenz zum Beispiel auf klassische, nicht modellbasierte Verfahren ausgewichen werden kann.

Das Modellbildungsverfahren aus Kapitel 5 kann explizit gelernte Regionen des Eingaberaumes repräsentieren und ein Konfidenzmaß zur Beurteilung der Modellvorhersage liefern.

3.4.3 Unvorhergesehenes Vergessen bei kontinuierlichem Lernen

Ein großer Nachteil der *globalen* Informationspeicherung in Neuronalen Netzen ist das unvorhersehbare Vergessen bereits gelernter Information. Das bedeutet, daß durch Präsentation neuer Information alte gelöscht oder – unter Umständen der schlimmere Fall – verfälscht wird. Wie in [31] sowohl für natürliche als auch für künstliche Neuronale Netze dargelegt wird, kann das soweit gehen, daß die komplette bisher gelernte Information verloren geht. Auch in [4] wird das Problem aus der Sicht der Regelungstechnik angesprochen und als schwerwiegend eingestuft. Nun gibt es einige Verfahren, die durch Mischen oder explizites Selektieren der Trainingsdaten diese Effekte minimieren. Dies läßt sich allerdings nur im Off-Line-Training realisieren und kommt somit im betrachteten Fall des kontinuierlichen Lernens nicht in Frage.

Aufgrund seiner lokalen Kartierung des Eingaberaumes ist das Verfahren aus Kapitel 5 in der Lage jeweils kleine *lokale* Modelle unterschiedlicher Struktur zu nutzen, so daß das globale Vergessen unterbunden wird.

Kapitel 4

Datenerfassung

Das folgende Kapitel befaßt sich mit der Vermessung technischer Prozesse und der Vorverarbeitung der erhaltenen Daten, um anschließend eine Modellierung des Systems vorzunehmen. Im Besonderen soll dabei auf die On-Line-Fähigkeit geachtet werden, um den Einsatz in Kombination mit dem in Kapitel 5 entwickelten Verfahren zu ermöglichen.

4.1 Filterung

Sollen physikalische Prozesse beschrieben werden, so erfolgt dies sehr oft durch Differentialgleichungssysteme. Das Problem besteht dabei häufig nicht etwa darin, den Prozeß grundsätzlich zu verstehen, sondern in der Parametrierung, d.h. Feinabstimmung, des Prozeßmodells. Zur exakten Parameterjustierung ist man auf eine Vermessung des realen Prozesses angewiesen. Treten Störungen, z.B. Rauschen, hochfrequente Einstreuungen, oder auch systematische Fehler, wie etwa hardwarebedingte Ausreißer oder Signalverluste auf, so kann dies durch geeignete Filter teilweise kompensiert werden.

Noch wichtiger ist ein exaktes Meßsignal, wenn es sich um eine Regelgröße handelt, aufgrund derer ein Regler seine Stellgröße justiert. Treten hier Ausreißer oder starkes Rauschen auf, die nicht durch den realen Prozeß, sondern durch Meßfehler induziert wurden, so wird der Regler völlig unnötig versuchen, dies auszuregeln.

4.1.1 Einführung verschiedener Filtertypen

In der Regel verfügt ein Filter über eine oder mehrere Grenzfrequenzen. Diese charakterisieren jeweils den Übergang vom sogenannten Durchlaß- in den Sperrbereich und umgekehrt. Dadurch werden Frequenzbänder definiert, die den Filter im Idealfall komplett bzw. gar nicht passieren können. Je nach Zielsetzung der Filterung unterscheidet man zwischen Tief-, Band- und Hochpassfiltern. Der Tief- bzw. Hochpassfilter verfügt über eine einzige Grenzfrequenz, so daß Frequenzen unter- bzw. oberhalb dieser herausgefiltert werden. Bandpassfilter hingegen lassen nur ein durch zwei Grenzfrequenzen definierten Frequenzbereich passieren.

Zusätzlich unterscheidet man, je nach Realisierung, zwischen digitalen und analogen Filtern. Im Computer bzw. Prozeßrechner als Algorithmus hinterlegte Filter werden als digital bezeichnet, da sie – im Gegensatz zu analog in Hardware aufgebauten Filtern – mit diskreter Abtastung arbeiten. Anhand der Impulsantwort[1] können zwei Arten von digitalen Filtern unterschieden werden [74]:

[1] Die Impulsantwort ist die Reaktion auf die Dirac-Funktion $\delta(t)$ als Eingabe. (Siehe auch Seite 6)

Finite-Impuls-Response-Filter (FIR)

Hierbei handelt es sich um Filter mit begrenzter Impulsantwortzeit. Der Effekt eines kurzen Impulses geht also nach einer für diesen Filter spezifischen Zeit nicht mehr in sein Verhalten ein. Dieser Typ von Filtern läßt sich ohne Rückkopplungen realisieren und ist somit immer stabil. Zusätzlich sind ein linearer Phasenverlauf sowie eine konstante Gruppenlaufzeit gewährleistet.

Infinite-Impuls-Response-Filter (IIR)

Um die für diesen Filter typische unendliche Impulsantwortzeit zu realisieren, ist eine Rückkopplung der Filterausgabe notwendig. Wie bei analogen Filtern auch, kann es dadurch unter bestimmten Bedingungen zu Schwingungen kommen, die die Stabilität des Systems beeinträchtigen. Speziell durch begrenzte Rechengenauigkeit – wie sie in Prozeßrechnern üblich ist – auftretende Rundungsfehler sind mit Rückkopplungszweig problematisch, da sich diese potenzieren. Der Vorteil dieses Filters gegenüber einem FIR-Filter liegt zum einen in der niedrigeren benötigten Ordnung um einen vergleichbaren Filtereffekt zu erreichen. Zum anderen bedingt dies einen wesentlich geringeren Rechenaufwand und Speicherplatzbedarf zur Laufzeit.

Im weiteren Verlauf soll nur noch auf die, in diesem Kontext interessanten, digitalen Tiefpassfilter eingegangen werden.

4.1.2 Beispiele klassischer IIR-Filter

In der Abbildung 4.1 wird der Frequenzgang einiger typischer Vertreter der IIR-Filter gezeigt. Dabei handelt es sich im Einzelnen um:

Elliptische-Filter zeichnen sich durch die niedrigste benötigte Systemordnung bei festgelegter Filterleistung aus. Ein besonderes Merkmal ist das steile Abfallen bei Erreichen der Grenzfrequenz. Der wellenförmige Verlauf des Frequenzgangs im Durchlaß- und Sperrbereich ist allerdings eine häufig unerwünschte Eigenschaft.

Butterworth-Filter bieten einen maximal flachen Frequenzgang im Durchlaßbereich. Der komplette Frequenzgang ist monoton, der zuvor bemängelte wellenförmige Verlauf ist nicht vorhanden. Diese Eigenschaften erkauft man sich allerdings mit einem nur mäßig steilen Abfallen im Bereich der Grenzfrequenz.

Chebyshev-„Typ I“-Filter besitzen einen maximal flachen Frequenzgang im Sperrbereich und ein wesentlich steileres Abfallen, als etwa der Butterworth Filter. Sie besitzen jedoch wieder einen wellenförmigen Verlauf im Durchlaßbereich des Frequenzgangs.

Chebyshev-„Typ II“-Filter werden auch oft als inverser Chebyshev-“Typ I“-Filter bezeichnet. Sie fallen bei Erreichen der Grenzfrequenz nicht ganz so steil ab, wie der „Typ I“-Filter, besitzen dafür aber einen maximal flachen Frequenzgang im Durchlaßbereich und einen wellenförmigen Verlauf im Sperrbereich, was häufig vorteilhaft ist.

Eine tiefergehende Analyse der vorgestellten IIR-Filter mit Angabe der möglichen Parametervariationen kann zum Beispiel in [102] gefunden werden.

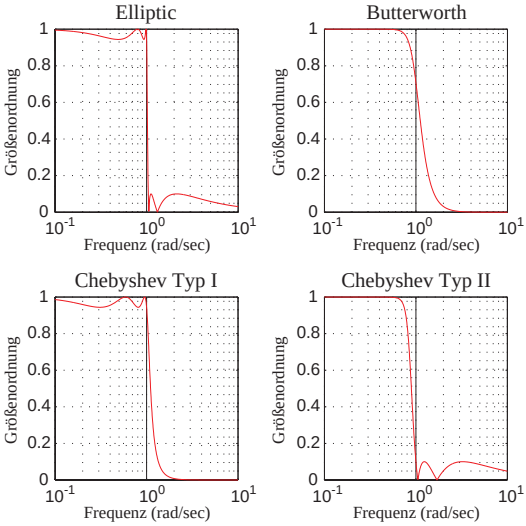


Abbildung 4.1: Vergleich des Frequenzgangs einiger klassischer IIR-Filter nach [102].

4.1.3 Modellbasierte FIR-Filter

Es existieren eine ganze Reihe verschiedener FIR-Filter, wie zum Beispiel in [44, 74] nachzulesen ist. Jackson [44] unterscheidet dabei zwischen speziellen Entwurfstechniken für IIR- bzw. FIR-Filter und sogenannten modellbasierten Entwurfstechniken, die jeweils ein bestimmtes lokales Modell voraussetzen. Dies führt in jedem Schritt der Filterung zu einem lokalen Modellbildungsproblem. Theoretisch kann somit jedes beliebige lokale Modell, sofern es adaptiv ist, verwendet werden. Praktisch erzwingt die in den Prozeßrechnern verfügbare Rechenleistung die Wahl einer „angemessenen“ Modellkomplexität. Es bietet sich an, möglichst einfache lokale Modelle zu wählen. Im nächsten Abschnitt wird nun eine Klasse von modellbasierten FIR-Filtern mit lokal polynomiellen Modellen hergeleitet.

Polynomielle Modelle

Die folgenden Herleitungen gelten für die Filterung eines mehrdimensionalen Meßsignals, das bedeutet, zu einem Zeitpunkt werden jeweils mehrere verschiedene Signale gleichzeitig gemessen. Sei nun das Meßsignal gegeben als $\{\{t_1, \mathrm{u}(t_1)\}, \ldots, \{t_n, \mathrm{u}(t_n)\}\}$, eine Menge von n Wertepaaren, die im weiteren kurz als $\{\{t_1, \vec{u}_1\}, \ldots, \{t_n, \vec{u}_n\}\}$ geschrieben wird. Dabei bezeichne t_i den Zeitpunkt der Messung und $\vec{u}_i$ die dabei erfaßten Meßwerte. Weiterhin gelte $t_i < t_{i+1}$, d.h. die Meßwerte sind zeitlich nacheinander erfaßt worden.

Will man ein Polynom $m$ter-Ordnung mittels Regression an die n Meßpunkte anpassen, so kann dieses Problem als lineares Ausgleichsproblem formuliert werden, wenn man die Matrizen $\mathbf{T}$ und $\mathbf{U}$ einführt:

$$\mathbf{U} = \begin{bmatrix} \vec{u}_1 & \cdots & \vec{u}_n \end{bmatrix}, \quad \mathbf{T} = \begin{bmatrix} 1 & 1 & & 1 \\ t_1 & t_2 & \cdots & t_n \\ \vdots & \vdots & & \vdots \\ t_1^m & t_2^m & & t_n^m \end{bmatrix}$$

Dabei enthält die Matrix $\mathbf{T}$ die Meßzeitpunkte in allen zum entsprechenden Polynomgrad gehörigen Potenzen. Die Matrix $\mathbf{U}$ entsteht aus den zugehörigen Meßpunkten, die sie als Spalten enthält. Das Regressionsproblem reduziert sich nun auf die Suche nach der Koeffizientenmatrix $\mathbf{A}$, die das lineare System $\mathbf{U} = \mathbf{AT}$ löst. Je nach Zahl n der Meßpunkte und Grad m des Polynoms existieren verschiedene Lösungen:

$\mathbf{n < m + 1}$: Das Gleichungssystem ist unterbestimmt, eine Lösung ist nicht sinnvoll. Es erscheint ratsam, in diesem Fall den Polynomgrad m auf $n - 1$ zu erniedrigen und somit eine exakte Lösung zu ermöglichen, wie es der nächste Fall darstellt.

$\mathbf{n \equiv m + 1}$: Das Gleichungssystem ist exakt lösbar, da $\mathbf{T}$ quadratisch ist und vollen Rang besitzt. Um den vollen Rang zu garantieren, dürfen verschiedene Messungen nur zu unterschiedlichen Zeitpunkten vorgenommen werden. Somit ist $\mathbf{T}$ invertierbar und die Koeffizientenmatrix $\mathbf{A} = \mathbf{UT}^{-1}$ ist die exakte Lösung.

$\mathbf{n > m + 1}$: Das Gleichungssystem ist überbestimmt und somit existiert keine exakte Lösung. Man kann jedoch eine eindeutige Koeffizientenmatrix $\mathbf{A}$ angeben, die den quadratischen Fehler $\|\mathbf{U} - \mathbf{AT}\|_{\text{Frobenius}}$ minimiert. Diese hat dann die Form $\mathbf{A} = \mathbf{UT}^{+}$. Die sogenannte Pseudoinverse $\mathbf{T}^{+} = \mathbf{T}^T(\mathbf{TT}^T)^{-1}$ von $\mathbf{T}$ stellt dabei die beste mögliche Näherung der Inversen von $\mathbf{T}$ dar. Beweise zur Form, Eindeutigkeit und zur beschriebenen Minimierungseigenschaft der Pseudoinversen können in [93] nachgelesen werden. Es bleibt noch zu erwähnen, daß für den Fall $n = m + 1$ die Inverse und die Pseudoinverse von $\mathbf{T}$ identisch sind.

Zur eigentlichen Berechnung der Koeffizientenmatrix $\mathbf{A}$ ist also eine Matrixmultiplikation und die Berechnung der Pseudoinversen $\mathbf{T}^{+}$ notwendig. Da eine direkte Berechnung von $\mathbf{T}^{+}$ über die Formel $\mathbf{T}^T(\mathbf{TT}^T)^{-1}$ aufgrund der notwendigen Matrixinvertierung numerisch instabil ist, existieren eine Reihe von Iterationsverfahren, um diese Berechnung numerisch stabil durchzuführen. Leider sind diese in der Regel mit einem erheblichen Rechenaufwand verbunden, so daß es sich lohnt über Alternativen nachzudenken: Im folgenden wird eine neuartige, durch geschickte Skalierung der Zeitachse *explizite*, Darstellung der Pseudoinversen $\mathbf{T}^{+}$ hergeleitet. Dadurch kann auf aufwendige Iterationsverfahren verzichtet werden.

Setzt man diskrete, äquidistante Meßzeitpunkte voraus, so läßt sich die Zeitachse umskalieren, so daß $\forall i \in \{1, \ldots, n\} : t_i = i$ gilt. Man erhält dann eine spezielle Form von $\mathbf{T}$, die im folgenden mit $\mathbf{T}_{m,n}$ bezeichnet wird:

$$\mathbf{T}_{m,n} = \begin{bmatrix} 1 & 1 & & 1 \\ 1 & 2 & & n \\ \vdots & \vdots & \cdots & \vdots \\ 1 & 2^m & & n^m \end{bmatrix} \tag{4.1}$$

Für diesen Spezialfall der äquidistanten polynomiellen Regression läßt sich die Pseudoinverse aus $\mathbf{T}^T(\mathbf{T}\mathbf{T}^T)^{-1}$ direkt analytisch berechnen. Aus Formel (4.1) folgt unmittelbar:

$$\mathbf{T}_{m,n}\mathbf{T}_{m,n}^T = \begin{bmatrix} s_{0,n} & \cdots & s_{m,n} \\ \vdots & & \vdots \\ s_{m,n} & \cdots & s_{2m,n} \end{bmatrix}, \quad \text{mit } s_{i,n} = \sum_{k=1}^{n} k^i \tag{4.2}$$

Für die Fälle $m \in \{0, 1, 2\}$ läßt sich diese Matrix einfach[2] analytisch invertieren. Dazu ersetzt man zuerst $s_{i,n}$ durch:

$$\begin{aligned} s_{0,n} &= \sum_{k=1}^{n} 1 = n \\ s_{1,n} &= \sum_{k=1}^{n} k = \frac{n(n+1)}{2} \\ s_{2,n} &= \sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6} \\ s_{3,n} &= \sum_{k=1}^{n} k^3 = \frac{n^2(n+1)^2}{4} \\ s_{4,n} &= \sum_{k=1}^{n} k^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30} \\ &\vdots \end{aligned} \tag{4.3}$$

Nach anschließender Invertierung und durch linksseitige Multiplikation mit $\mathbf{T}_{m,n}^T$ ergeben sich schließlich die Pseudoinversen $\mathbf{T}_{\{0,1,2\},n}^+$ zu:

$$\begin{aligned} \mathbf{T}_{0,n}^+ &= \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \cdot \begin{bmatrix} \frac{1}{n} \end{bmatrix} \\ \mathbf{T}_{1,n}^+ &= \begin{bmatrix} 1 & 1 \\ \vdots & \vdots \\ 1 & n \end{bmatrix} \cdot \begin{bmatrix} \frac{2(2n+1)}{n(n-1)} & \frac{6}{n(1-n)} \\ \frac{6}{n(1-n)} & \frac{12}{n(n-1)(n+1)} \end{bmatrix} \\ \mathbf{T}_{2,n}^+ &= \begin{bmatrix} 1 & 1 & 1 \\ \vdots & \vdots & \vdots \\ 1 & n & n^2 \end{bmatrix} \cdot \begin{bmatrix} \frac{3(3n^2+3n+2)}{n(n-1)(n-2)} & \frac{-18(2n+1)}{n(n-1)(n-2)} & \frac{30}{n(n-1)(n-2)} \\ \frac{-18(2n+1)}{n(n-1)(n-2)} & \frac{12(8n+11)(2n+1)}{n(n-1)(n-2)(n+1)(n+2)} & \frac{-180}{n(n-1)(n-2)(n+2)} \\ \frac{30}{n(n-1)(n-2)} & \frac{-180}{n(n-1)(n-2)(n+2)} & \frac{180}{n(n-1)(n-2)(n+1)(n+2)} \end{bmatrix} \end{aligned} \tag{4.4}$$

Da man nun über die Koeffizientenmatrix $\mathbf{A} = \mathbf{U}\mathbf{T}^+$ verfügt, können beliebige Punkte t des Polynoms berechnet werden. Hierzu multipliziert man einfach die Koeffizientenmatrix mit dem Polynomvektor des Zeitpunktes t und erhält folgende Funktion:

[2]Natürlich lassen sich auch für $m > 2$ entsprechende Matrixinvertierungen finden, allerdings erscheint es im Kontext der lokalen Modelle zur Filterung nicht sinnvoll, Polynome höherer Ordnung einzusetzen.

$$\hat{\mathrm{u}} : t \mapsto \mathbf{U}\mathbf{T}^{+}_{m,n} \cdot \begin{bmatrix} 1 \\ t \\ \vdots \\ t^m \end{bmatrix} \tag{4.5}$$

Überprüft man nun die Filterleistung anhand des Frequenzgangs, so stellt sich heraus, daß die eben entwickelte Form der Regressionsfilter über einen stark wellenförmigen Verlauf im Sperrbereich verfügen, wie es in Abbildung 4.2 zu sehen ist. Dies ist, wie etwa in [44] nachgelesen werden kann, auf das abrupte Abbrechen der zur Filterung verwendeten Datenpunkte nach n Schritten zurückzuführen. Üblicherweise wird zu Vermeidung dieses Effekts eine Gewichtungsfunktion verwendet, die weiter zurückliegenden Samples eine niedrigere Priorität bei der Filterung zuweist. Eine gängige Gewichtungsfunktion ist zum Beispiel die Gaußsche Glockenfunktion. In die Regressionsfilter kann eine Gewichtungsfunktion $\mathrm{g}(t)$, die den verschiedenen Meßzeitpunkten t_i verschiedene Gewichte $\mathrm{g}(t_i)$ zuordnet, mit Hilfe der Diagonalmatrix $\mathbf{G}$ mit Eigenwerten $\{\mathrm{g}(t_1), \ldots, \mathrm{g}(t_n)\}$ eingebracht werden. Dazu wird nochmals das lineare Ausgleichsproblem betrachtet, dessen Lösung einer Minimierung das quadratischen Fehlers $\|\mathbf{U} - \mathbf{A}\mathbf{T}\|_{\text{Frobenius}}$ entspricht. Die modifizierte Form mit integrierter Gewichtungsfunktion lautet nun:

$$\|(\mathbf{U} - \tilde{\mathbf{A}}\mathbf{T})\mathbf{G}\|_{\text{Frobenius}} = \|\mathbf{U}\mathbf{G} - \tilde{\mathbf{A}}\mathbf{T}\mathbf{G}\|_{\text{Frobenius}} \tag{4.6}$$

Dies entspricht der Lösung des linearen Systems $\mathbf{U}\mathbf{G} = \tilde{\mathbf{A}}\mathbf{T}\mathbf{G}$, die durch $\tilde{\mathbf{A}} = \mathbf{U}\mathbf{G}(\mathbf{T}\mathbf{G})^{+}$ gegeben ist. Somit lautet Gleichung (4.5) nun:

$$\hat{\mathrm{u}} : t \mapsto \mathbf{U} \underbrace{\mathbf{G}(\mathbf{T}_{m,n}\mathbf{G})^{+}}_{\tilde{\mathbf{T}}^{+}_{m,n}} \cdot \begin{bmatrix} 1 \\ t \\ \vdots \\ t^m \end{bmatrix} \tag{4.7}$$

Gesucht ist demnach nicht mehr $\mathbf{T}^{+}_{m,n}$, sondern $\mathbf{G}(\mathbf{T}_{m,n}\mathbf{G})^{+}$, im folgenden kurz mit $\tilde{\mathbf{T}}^{+}_{m,n}$ bezeichnet[3]. Nun wird auch das eigentliche Problem dieser neuen Form deutlich: Man kann die Berechnung der Pseudoinversen von $\mathbf{T}_{m,n}\mathbf{G}$ *nicht* auf die Berechnung der einzelnen Pseudoinversen von $\mathbf{T}_{m,n}$ und $\mathbf{G}$ zurückführen, denn es gilt:

$$(\mathbf{T}_{m,n}\mathbf{G})^{+} = \mathbf{G}\mathbf{T}^{T}_{m,n}(\mathbf{T}_{m,n}\mathbf{G}^2\mathbf{T}^{T}_{m,n})^{-1}$$

Somit kann nicht einfach irgend eine beliebige Gewichtungsfunktion genutzt werden. Es läßt sich jedoch eine lineare Gewichtungsfunktion[4] finden, deren Anwendung wieder zu einer analytisch lösbaren Form von $(\mathbf{T}_{m,n}\mathbf{G})^{+}$ führt. Diese lautet:

$$\mathrm{g} : t \mapsto \frac{t}{n} \tag{4.8}$$

[3] Bei $\tilde{\mathbf{T}}^{+}_{m,n}$ handelt es sich *nicht* um eine Pseudoinverse, sondern um eine Abkürzung.

[4] Natürlich existieren noch andere Gewichtungsfunktionen, die zu einer analytisch lösbaren Form führen.

Dies bedeutet, daß der letzte Datenpunkt mit der Gewichtung Eins, der vorletzte mit der Gewichtung $\frac{n-1}{n}$, usw. eingeht. Da sich skalare Faktoren in $\mathbf{G}$ nicht auf die Lösung von Gleichung (4.6) auswirken, kann anstelle von $g(t)$ auch $g(t) \cdot n$ verwendet werden. Man erhält somit:

$$\mathbf{T}_{m,n}\mathbf{G} = \begin{bmatrix} 1 & 2 & & n \\ 1 & 4 & & n^2 \\ \vdots & \vdots & \cdots & \vdots \\ 1 & 2^{m+1} & & n^{m+1} \end{bmatrix} \quad \Longrightarrow \quad \mathbf{T}_{m,n}\mathbf{G}^2\mathbf{T}_{m,n}^T = \begin{bmatrix} s_{2,n} & \cdots & s_{m+2,n} \\ \vdots & & \vdots \\ s_{m+2,n} & \cdots & s_{2m+2,n} \end{bmatrix} \tag{4.9}$$

Nach anschließender Invertierung und durch linksseitige Multiplikation mit $\mathbf{G}\mathbf{T}_{m,n}^T$ ergibt sich schließlich $\tilde{\mathbf{T}}^+_{\{0,1,2\},n}$ zu:

$$\begin{aligned}
\tilde{\mathbf{T}}^+_{0,n} &= \begin{bmatrix} 1 \\ \vdots \\ n^2 \end{bmatrix} \cdot \left[\frac{6}{n(n+1)(2n+1)} \right] \\
\tilde{\mathbf{T}}^+_{1,n} &= \begin{bmatrix} 1 & 1 \\ \vdots & \vdots \\ n^2 & n^3 \end{bmatrix} \cdot \begin{bmatrix} \frac{24(3n^2+3n-1)(2n+1)}{n(n-1)(n+2)(n+1)(3n^2+3n+2)} & \frac{-180}{(n-1)(n+2)(n+1)(3n^2+3n+2)} \\ \frac{-180}{(n-1)(n+2)(3n^2+3n+2)} & \frac{120(2n+1)}{n(n-1)(n+2)(n+1)(3n^2+3n+2)} \end{bmatrix} \\
\tilde{\mathbf{T}}^+_{2,n} &= \begin{bmatrix} 1 & 1 & 1 \\ \vdots & \vdots & \vdots \\ n^2 & n^3 & n^4 \end{bmatrix} \cdot \frac{1}{(n-1)(n-2)(n+3)(n+2)(n^2+n+3)} \cdot \\
&\quad \begin{bmatrix} \frac{150(4n^6+12n^5+4n^4-12n^3-11n^2-3n+2)}{n(2n+1)(n+1)} & -300(3n^2+3n-4) & \frac{210(6n^4+12n^3+3n^2-3n+2)}{n(2n+1)(n+1)} \\ -300(3n^2+3n-4) & \frac{360(2n+3)(2n+1)(2n-1)}{n(n+1)} & -2100 \\ \frac{210(6n^4+12n^3+3n^2-3n+2)}{n(2n+1)(n+1)} & -2100 & \frac{1050(3n^2+3n+2)}{n(2n+1)(n+1)} \end{bmatrix}
\end{aligned} \tag{4.10}$$

In Abbildung 4.2 ist nun der Frequenzgang des Regressionsfilters mit und ohne lineare Gewichtung gezeigt. Wie erwartet, ist der wellenförmige Verlauf im Sperrbereich verschwunden. Man erkauft sich diese unter Umständen notwendige Eigenschaft allerdings mit einem etwas flacheren Übergang vom Durchlaß- in den Sperrbereich.

Mit Hilfe der Darstellung in Gleichung (4.4) und Gleichung (4.10) ist nun eine numerisch stabile und effizient zu berechnende Form des Regressionsfilters verfügbar. Dies ist speziell für Anwendungen auf Prozeßrechnern mit beschränktem Speicherplatz und beschränkter Rechenkapazität von Interesse, da auf die für eine numerisch stabile Berechnung der Pseudoinversen generell notwendige Singuläre-Werte-Zerlegung verzichtet werden kann.

4.1.4 Probleme durch Ausreißer und Sprünge

Im folgenden soll nun exemplarisch das Verhalten einiger IIR-Filter bei den bereits beschriebene Problempunkten Ausreißer und Sprünge dargestellt werden. Abbildung 4.3 zeigt die Reaktion klassischer IIR-Filter auf den Einheitssprung[5]. Das verspätete Ansteigen und anschließende Über- bzw. Einschwingen auf den neuen Signalwert ist symptomatisch für lineare Systeme. Nachdem der Einheitssprung im Frequenzraum einer Mischung *aller* Frequenzen entspricht, ist es offensichtlich, daß durch Tiefpassfiltern genau der Frequenzanteil entfernt wird, der für den schnellen Anstieg auf das neue Signalniveau zuständig wäre. Je niedriger die Grenzfrequenz des verwendeten Filters liegt, desto langsamer erfolgt der Anstieg beim Einheitssprung.

[5]Dies ist das sprunghafte Ansteigen des Eingabewertes von Null auf Eins, mit anschließendem Halten des Wertes Eins. (Siehe auch Seite 6)

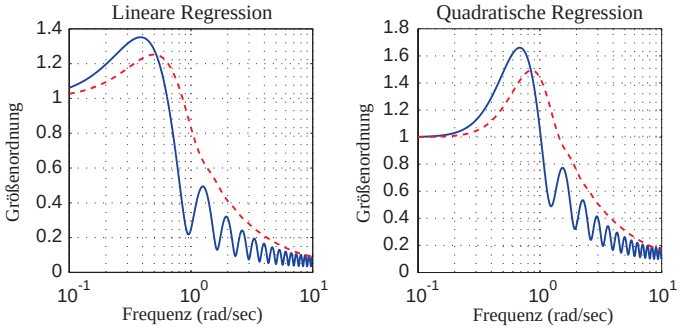


Abbildung 4.2: Hier sind die Frequenzgänge verschiedener Regressionsfilter zu sehen. Blau durchgezogen ist dabei der Regressionsfilter ohne, rot gestrichelt mit linearer Gewichtung eingezeichnet. Links wurde ein lineares, rechts ein quadratisches Regressionsmodell verwendet. Man erkennt deutlich den wellenförmigen Verlauf des Frequenzgangs beim Regressionsfilter ohne lineare Gewichtung. Dieser Effekt wird, wie aus der Grafik ersichtlich ist, durch die lineare Gewichtung vermieden.

In der Regel muß man also einen Kompromiß zwischen schnellem Anstieg und einem damit gekoppelten starkem Überschwingen oder ein langsamen Anstieg mit minimalem Überschwingen finden. Zusätzlich erschwert wird die Problematik durch die Existenz von Ausreißern. Versieht man den Filter mit einer höheren Grenzfrequenz, um Sprüngen schneller zu folgen, so haben auch einzelne Ausreißer automatisch eine stärkere Auswirkung auf den Filterausgang.

Um die Reaktion auf Sprünge der Orginaldynamik zu verbessern, wird im nächsten Abschnitt ein Varianztest entwickelt, der besonders einfach in Kombination mit Regressionsfiltern eingesetzt werden kann.

4.1.5 Sprungerkennung anhand von Varianzkriterien

Will man Signalsprünge erkennen und von Ausreißern unterscheiden, so stellt sich automatisch die Frage nach der Signaldynamik. Man muß also die Eigendynamik des Signals feststellen, und darauf basierend entscheiden, ob ein Meßwert als Ausreißer oder wirklicher Sprung einzustufen ist. Die weiteren Überlegungen werden unter der Voraussetzung angestellt, daß sich das gemessene Signal $\tilde{\mathrm{f}}(t)$ als additive Überlagerung des Nutzsignals $\mathrm{f}(t)$ und des Rauschens $\xi(t)$ zusammensetzt und somit $\tilde{\mathrm{f}}(t) = \mathrm{f}(t) + \xi(t)$ gilt. Desweiteren sollen $\mathrm{f}(t)$ und $\xi(t)$ unabhängig sein.

Dämpft bzw. eliminiert der verwendete Filter $\mathfrak{F}$ das Rauschen, läßt jedoch das Nutzsignal ungehindert passieren, so erhält man als Schätzung[6] des tatsächlich auftretenden Rauschens:

$$\hat{\xi}(t) \approx \tilde{\mathrm{f}}(t) - \mathfrak{F}[\tilde{\mathrm{f}}(t)]$$

Setzt man nun eine bestimmte Dynamik des Rauschen $\xi(t)$ voraus, so lassen sich aus der Schätzung $\hat{\xi}(t)$ einige Kenngrößen berechnen, anhand derer Sprünge im Nutzsignal vom Rauschen und gelegentlichen Ausreißern getrennt werden können. Im weiteren Verlauf wird nun

[6] Im weiteren Verlauf werden Schätzungen mit ˆ markiert.

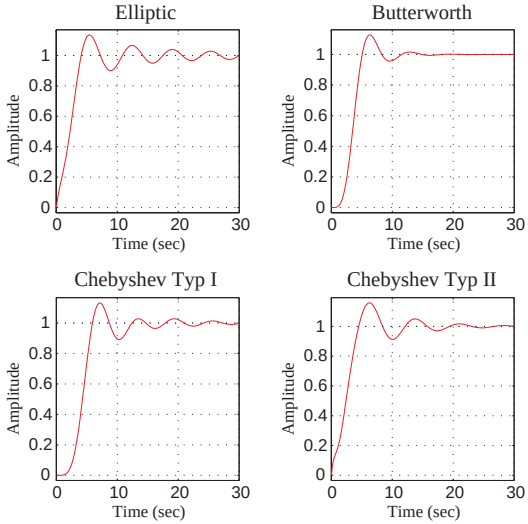


Abbildung 4.3: Vergleich der Sprungantwort einiger klassischer IIR-Filter nach [102]. Typisch ist das verspätete Ansteigen und anschließende Über- bzw. Einschwingen als Reaktion auf den Sprung zum Zeitpunkt Null.

ein normalverteiltes Rauschen mit Varianz σ^2 und Mittelwert μ angenommen[7]. Die Dichtefunktion $\phi_{\mu,\sigma}$ und Wahrscheinlichkeitsverteilung $\Phi_{\mu,\sigma}$ der Normalverteilung mit Mittelwert μ und Varianz σ^2 lauten:

$$\phi_{\mu,\sigma}(x) = \frac{1}{\sigma\sqrt{2\pi}}\, e^{-\frac{(x-\mu)^2}{2\sigma^2}} \qquad \Phi_{\mu,\sigma}(x) = \int_{-\infty}^{x} \phi_{\mu,\sigma}(t)dt \tag{4.11}$$

Mit Hilfe statistischer Schätzer kann aus den k letzten Messungen der Mittelwert μ und die Varianz σ^2 des Rauschens geschätzt werden. Die normal verwendeten Schätzer sind [6]:

$$\hat{\mu} = \frac{1}{k}\sum_{i=1}^{k} \hat{\xi}(t-i+1) \qquad \hat{\sigma}^2 = \frac{1}{k-1}\sum_{i=1}^{k} (\hat{\xi}(t-i+1) - \hat{\mu}) \tag{4.12}$$

Durch die Schätzwerte für μ und σ kann die Wahrscheinlichkeit, daß sich das Rauschen $\xi(t)$ in einem Wertebereich $[\mu - \delta\sigma, \mu + \delta\sigma]$ befindet, folgendermaßen angegeben werden:

$$P(\xi(t) \in [\mu - \delta\sigma, \mu + \delta\sigma]) = \Phi_{\mu,\sigma}(\mu + \delta\sigma) - \Phi_{\mu,\sigma}(\mu - \delta\sigma) = \underbrace{\Phi_{0,1}(\delta) - \Phi_{0,1}(-\delta)}_{\tilde{\Phi}_{0,1}(\delta)}$$

Kennt man die Umkehrabbildung von $\tilde{\Phi}_{0,1}(x)$, so ist es möglich die Grenzen festzulegen, in denen sich das Rauschen mit einer festgelegten Wahrscheinlichkeit bewegt. In Abbildung 4.4

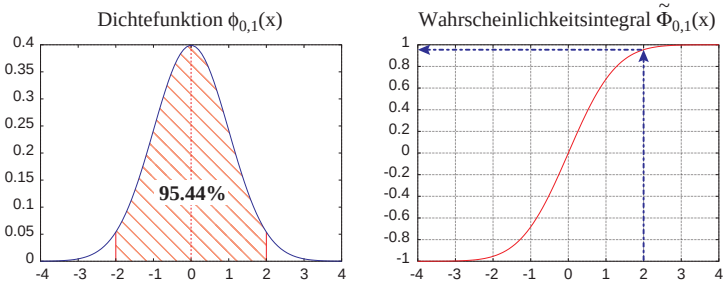


Abbildung 4.4: Im linken Teil ist als blau durchgezogene Linie die Dichtefunktion $\phi_{0,1}(x)$ der Normalverteilung zu sehen. Im schraffierten – um Null symmetrischen – Bereich befinden sich zum Beispiel 95.44% aller zufällig mit dieser Verteilung gezogenen Werte. Im rechten Teil ist das Wahrscheinlichkeitsintegral $\tilde{\Phi}_{0,1}(x)$ aus Formel (4.14) als rot durchgezogene Linie zu sehen. Es liefert die Fläche des links markierten Bereichs, indem man seine Grenze (Zwei) einsetzt, wie es mit den blau gestrichelten Pfeilen engedeutet ist. Folgt man den Pfeilen in umgekehrter Richtung, so erhält man zu einer gegebenen Wahrscheinlichkeit die Grenzen des zugehörigen Intervalls im linken Bild.

ist erläutert, wie in diesem Fall vorgegangen wird.

Bekanntermaßen läßt sich das Integral $\Phi_{\mu,\sigma}(x)$ nicht analytisch lösen. Es existieren umfangreiche Tabellen, aus denen die entsprechenden Werte entnommen werden können. Auf den meisten Computern steht darüberhinaus eine numerische Näherung der Funktion $\mathrm{erf}(x)$ zur Verfügung, aus der man folgendermaßen die Funktion $\Phi_{0,1}(x)$ erhalten kann:

$$\mathrm{erf}(x) = \frac{1}{\sqrt{\pi}} \int_{-x}^{x} e^{-t^2} dt \quad \overset{(4.11)}{\Longrightarrow} \quad \Phi_{0,1}(x) = \frac{1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)}{2} \tag{4.13}$$

Aus der ebenfalls verfügbaren numerischen Näherung der Umkehrabbildung $\mathrm{erf}^{(-1)}(y)$ läßt sich schließlich $\tilde{\Phi}_{0,1}^{(-1)}(y)$ berechnen:

$$\begin{aligned} \tilde{\Phi}_{0,1}(x) = 2(\Phi_{0,1}(x) - \Phi_{0,1}(0)) &\overset{(4.13)}{=} \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right) \\ \Downarrow & \\ \tilde{\Phi}_{0,1}^{(-1)}(y) = \sqrt{2}\mathrm{erf}^{(-1)}(y) & \end{aligned} \tag{4.14}$$

Somit läßt sich zu einer vorgegebenen Wahrscheinlichkeit p das zugehörige Aufenthaltsintervall berechnen. Es hat die Form $[\mu - \sigma \cdot \tilde{\Phi}_{0,1}^{(-1)}(p), \mu + \sigma \cdot \tilde{\Phi}_{0,1}^{(-1)}(p)]$. Durch Wahl der Wahrscheinlichkeit p legt man die akzeptierten Grenzen des Rauschens $\xi(t)$ fest, so daß ein Überschreiten der Grenzen als Sprung des Nutzsignals gedeutet werden kann. Um sich vor einzelnen Ausreißern zu schützen, kann das Überschreiten der festgesetzten Grenzen nicht nur anhand einzelner Datenpunkte des Rauschens, sondern auch am Mittelwert einer bestimmten

[7] Für andere Verteilungen kann analog zur beschriebenen Vorgehensweise eine Sprungerkennung durchgeführt werden.

Zahl von Punkten gemessen werden. Allgemein lautet die Bedingung für einen Sprung des Nutzsignals zum Zeitpunkt t also:

$$|\xi(t) - \mu| > \underbrace{\sigma \cdot \tilde{\Phi}_{0,1}^{(-1)}(p)}_{\text{Varianzgrenze}} \tag{4.15}$$

Alternativen zu obiger Vorgehensweise, wie etwa die Nutzung des Student-t-Tests, um den Mittelwert des Rauschens auf Änderungen zu testen, haben sich bei Versuchen an realen Daten als zu sensibel erwiesen, da Sprünge erkannt wurden, obwohl keine vorhanden waren. Im nächsten Abschnitt wird nun die Kombination der Sprungerkennung mit Hilfe des Varianztests und des zuvor vorgestellten Regressionsfilters erläutert.

4.1.6 Regressionsfilter mit Sprungerkennung

Die Vorteile der Kopplung der Regressionsfilter mit der zuvor eingeführten Sprungerkennung werden deutlich, wenn man sich nochmals die Reaktion eines Tiefpassfilters auf einen Sprung vor Augen hält: Es erfolgt ein mehr oder weniger langsamer Anstieg bis das Niveau der neuen[8] Dynamik erreicht ist. Wünschenswert wäre nun aber ein direkter Wechsel von der alten zur neuen Dynamik. Dies läßt sich mit klassischen IIR-Filtern wenn überhaupt, dann nur über Umwege – wie zum Beispiel über das explizite Berechnen der notwendigen integralen Anteile der Filterdynamik – realisieren, da diese theoretisch ja unendlich lange Zeit benötigen, bis die alte Dynamik „vergessen“ ist. Allein aus diesem Grund bietet sich die Verwendung von FIR-Filtern an, die nur ein Zeitfenster der Größe n berücksichtigen.

Nutzt man nun die vorgestellten Regressionsfilter, so kann man nach Erkennen eines Sprunges einfach die komplette Regressionsgeschichte löschen und somit ein explizites „Vergessen“ der alten Dynamik einleiten. Der Regressionsfilter wird also zuerst unmittelbar der neuen Dynamik *und* dem Rauschen folgen, bis der Filtereffekt mit zunehmender Zahl der Meßpunkte wieder einsetzt. Genau n Zeitschritte nach einem Sprung arbeitet der Filter wieder mit der gewünschten Filterleistung. Abbildung 4.5 zeigt die Sprungantwort eines solchen Filters im Vergleich zur Sprungantwort der Regressionsfilter ohne Sprungerkennung.

Da unmittelbar nach einem Dynamikwechsel noch nicht ausreichend viele Meßpunkte zur Verfügung stehen, um eine Sprungerkennung mit Hilfe des Varianztests durchzuführen, erfolgt eine erneute Sprungerkennung frühestens k Zeitschritte nach einem Sprung, so daß die gewünschte Zahl von Meßpunkten zum Varianztest verfügbar ist.

Als Nebenprodukt des Varianztests kann die Klassifikation von Ausreißern betrachtet werden. Übersteigt das Rauschen nur kurzzeitig die Varianzgrenze und wird somit *kein* Sprung erkannt, so können diese Werte als Ausreißer klassifiziert werden. Nachdem Ausreißer keinen Beitrag zur Signaldynamik leisten, können diese ignoriert werden, indem sie in der Regressionsgeschichte durch die korrespondierende Ausgabe des Filters ersetzt werden.

Abbildung 4.6 zeigt nochmals im Detail, wie die Sprungerkennung in Kombination mit den Regressionsfiltern funktioniert.

[8] Im folgenden wird von einem Dynamikwechsel durch den Sprung ausgegangen. Die beiden Dynamiken vor und nach dem Sprung werden mit *alter* und *neuer* Dynamik bezeichnet.

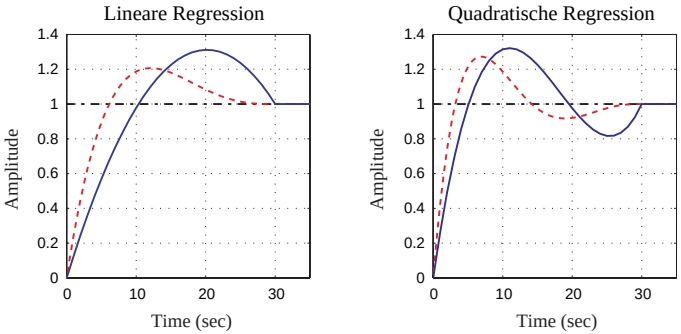


Abbildung 4.5: Hier sind die Sprungantworten derselben Regressionsfilter zu sehen, deren Frequenzgänge bereits in Abbildung 4.2 gezeigt wurden. Blau durchgezogen ist dabei der Regressionsfilter ohne, rot gestrichelt mit linearer Gewichtung eingezeichnet. Links wurde ein lineares, rechts ein quadratisches Regressionsmodell verwendet. Zusätzlich ist nun schwarz gestrichelt mit Punkten ein Regressionsfilter mit Sprungerkennung eingezeichnet. Wie zu erwarten, folgt sein Verlauf unmittelbar der Treppenfunktion und zeigt keine Verzögerung wie die Filter ohne Sprungerkennung oder die in Abbildung 4.3 gezeigten klassischen Filter.

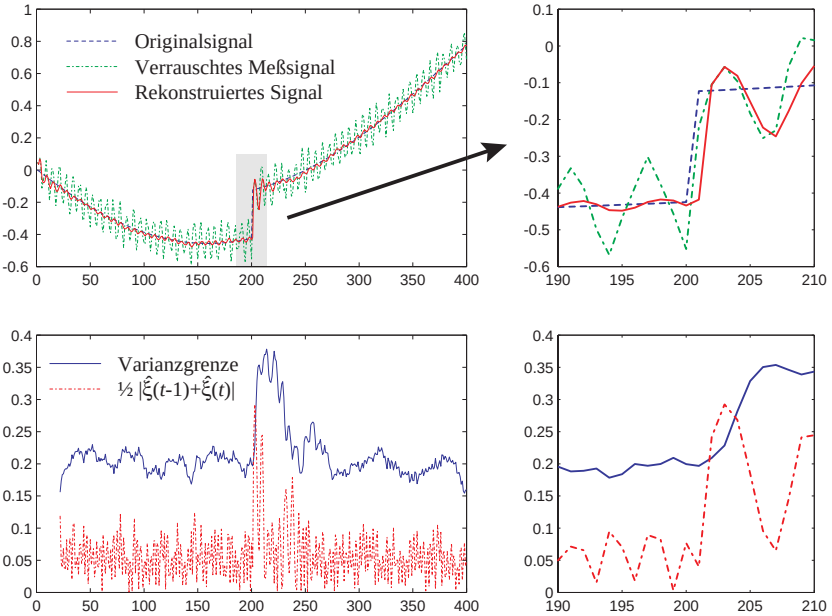


Abbildung 4.6: Beispiel für die Funktionsweise des Regressionsfilters mit Sprungerkennung. Im oberen Teil ist blau gestrichelt das Originalsignal, grün gepunktet das verrauschte Meßsignal und rot durchgezogen das durch den Filter rekonstruierte Signal mit einem Sprung bei $t = 201$ zu sehen. Im unteren Teil ist rot gestrichelt der Betrag des mittleren geschätzten Rauschens $\frac{1}{2}|\hat{\xi}(t-1) + \hat{\xi}(t)|$ über zwei Zeitschritte und blau durchgezogen die für $p = 90\%$ errechnete Varianzgrenze aus Formel (4.15) eingezeichnet. Jeweils links ist der komplette Signalverlauf und rechts ein Ausschnitt der Sprungregion zu sehen. Betrachtet man den Verlauf der errechneten Varianzgrenze, so erkennt man deutlich, daß sich das gemittelte Rauschen in der Regel innerhalb der gewählten Varianzgrenze befindet. Erst zum Zeitpunkt $t = 202$, also bereits einen Zeitschritt *nach* dem Sprung im Originalsignal steigt der Wert über die Varianzgrenze und ein Sprung im verrauschten Meßsignal wird erkannt. Die bisherige Vergangenheitsinformation des Regressionsfilters wird gelöscht, und er folgt – wie im oberen vergrößerten Ausschnitt sichtbar – unmittelbar dem Sprung. Bereits nach wenigen Zeitschritten setzt die Filterwirkung des Regressionsfilters wieder ein, ein erneuter Varianztest erfolgt allerdings erst, wenn die gewünschte Zahl von Meßpunkten wieder zur Verfügung steht.

4.2 Auffinden optimaler Meßpunkte

Üblicherweise wird zur Modellierung eines physikalischen Prozesses ein Datensatz erstellt, der als Eingabe für eines der vorgestellten Modellierungsverfahren dient. Die Erstellung des Datensatzes erfolgt normalerweise durch Anfahren verschiedener Arbeitspunkte im Prozeßzustandsraum. Häufig wird bei der Auswahl der zu vermessenden Arbeitspunkte die gewählte Modellstruktur nicht berücksichtigt.

4.2.1 Kostenfunktion

Unter der Voraussetzung, daß die gewählte Modellstruktur den physikalischen Prozeß approximieren kann, ist es möglich die zu vermessenden Arbeitspunkte so zu wählen, daß ein maximaler Gewinn bezüglich einer *modellabhängigen* Kostenfunktionen erzielt wird [27].

Meist ist diese Kostenfunktion so gewählt, daß der Informationsgewinn durch die Messungen maximiert wird [20]. Dadurch kann der Zustandsraum mit einer minimalen Anzahl von Messungen erfaßt werden. Dies ist bei hochdimensionalen Zustandsräumen und teuren Messungen von großem wirtschaftlichem Interesse.

4.2.2 Ansätze

Ein übliches Optimalitätskriterium nach Hardin und Sloane [37] ist es, das Integral über die Varianz der Modellvorhersage bezüglich einer zu bewertenden Region R durch Auswahl einer sogenannten I-optimalen Sequenz vom n Meßpunkten $\{\{\vec{u}_1, \vec{y}_1\}, \ldots \{\vec{u}_n, \vec{y}_n\}\}$ zu minimieren. Da es nur für einfache Probleme möglich ist, diese komplexe Aufgabe zu lösen, schlägt Cohn [16] ein inkrementelles Verfahren vor. In jedem Lernschritt soll dabei die Änderung der Varianz der Modellausgabe bezüglich eines beliebigen Meßpunktes berechnet, und anschließend der Meßpunkt $\hat{\vec{u}}$ so gewählt werden, daß er dieses Integral minimiert. Die erwartete Änderung der Varianz für Modelle mit Ausgabedimension Eins lautet:

$$\Delta\mathrm{var}(\hat{\vec{u}}) = \int_{\vec{u}\in R} \frac{\mathrm{cov}_{\mathrm{f}_{\vec{p}}}(\vec{u}, \hat{\vec{u}})^2}{1 + \mathrm{var}_{\mathrm{f}_{\vec{p}}}(\vec{u})}\, d\vec{u}, \quad \text{mit} \quad \mathrm{var}_{\mathrm{f}_{\vec{p}}}(\vec{u}) \equiv \mathrm{cov}_{\mathrm{f}_{\vec{p}}}(\vec{u}, \vec{u}) \tag{4.16}$$

Die dabei verwendete Kovarianz $\mathrm{cov}_{\mathrm{f}_{\vec{p}}}(\vec{u}_k, \vec{u}_l)$ enthält Information über die bisher gelernten m Datenpunkte $\{\{\vec{u}_1, \vec{y}_1\}, \ldots \{\vec{u}_m, \vec{y}_m\}\}$ und lautet in der in Kapitel 3 eingeführten Terminologie:

$$\mathrm{cov}_{\mathrm{f}_{\vec{p}}}(\vec{u}_k, \vec{u}_l) = \left(\frac{\partial \mathrm{f}_{\vec{p}}(\vec{u}_k)}{\partial \vec{p}}\right)^T \left(\frac{\partial^2 \sum_{i=1}^{m} \mathrm{E}(\mathrm{f}_{\vec{p}}(\vec{u}_i), \vec{y}_i)}{\partial \vec{p}^{\,2}}\right)^{-1} \left(\frac{\partial \mathrm{f}_{\vec{p}}(\vec{u}_l)}{\partial \vec{p}}\right) \tag{4.17}$$

Neben den I-optimalen Sequenzen existieren noch ein Reihe weiterer Kriterien [37], die ebenfalls darauf basieren, daß *eine globale* Modellfunktion vorhanden ist. Für Polynome niedriger Ordnung ist in der Software `Gosset` von Hardin und Sloane [38] ein Verfahren realisiert, das unter anderem auch I-optimale Sequenzen von Meßwerten liefern kann.

Einen anderen Ansatz verfolgt Schmidhuber [87, 88] mit dem Prinzip der „adaptiven Neugier“. Hier wird eine Schätzung der Modellkonfidenz dazu genutzt „interessante“ Regionen des Zustandsraumes zu erforschen. Interessant bedeutet hier vereinfacht ausgedrückt, daß die Modellkonfidenz in diesem Bereich niedrig ist.

4.2.3 Probleme

Untersucht man obige Verfahren auf ihre Tauglichkeit hin, auch komplexere Aufgaben zu lösen, so stellen sich folgende Schwachpunkte heraus:

- Das Integral in Gleichung (4.16) ist für mehrdimensionale Modellregionen R nur mit erheblichem Aufwand berechenbar. Als Abhilfe werden Monte-Carlo-Methoden, wie Gibbs-Sampling, genannt. Allerdings sind auch diese aufwendig und es ist darüberhinaus fraglich, ob die bereits genäherte Gleichung (4.16) dann noch eine sinnvolle Aussage über $\Delta\mathrm{var}(\hat{\vec{u}})$ machen kann.

- Häufig ist es nicht möglich die Prozeßdynamik durch ein einziges, globales Modell zu approximieren, wie in Abschnitt 5.3 erläutert wird. Eine Analyse von Formel (4.17) zeigt, daß eine Aussage nur über Punkte $\vec{u}$ gemacht werden kann, die sich im Einzugbereich eines lokalen Modells befinden, welches bereits mindestens einen Punkt $\vec{u}_i$ gelernt hat. Ansonsten ist der Covarianzterm identisch zu Null. Dies gilt nicht nur für lokale Modelle, die über getrennte Einzugsbereiche verfügen, sondern auch für bedingt lokale Modelle wie etwa RBF-Netze, bei denen eine Überlappung der einzelnen Einzugsregionen existiert.

- Die „Adaptive Neugier“ garantiert nicht die optimale Meßpunktfolge. Generell kann auch hier keine Aussage über Regionen gemacht werden, in denen noch keine Punkte erfaßt wurden, sobald lokale Modelle genutzt werden.

Nach Abwägung der Vor- und Nachteile der diskutierten Verfahren, bietet sich die Verwendung nur bei gleichzeitiger Nutzung globaler Modellierungsverfahren und entsprechendem Vorwissen über den Prozeß an. Bei komplexeren Black-Box-Modellen, die mit Hilfe lokaler Modelle arbeiten, bietet sich eher das im nächsten Kapitel vorgestellte Verfahren an.

Kapitel 5

LEMON

Ein On-Line-Verfahren zur lokalen Modellbildung

Sollen neuronale Netze zur Modellierung physikalischer Prozesse eingesetzt werden, so kommt man schnell zu der Erkenntnis, daß wirkliches On-Line-Training bei fast allen herkömmlichen Architekturen in befriedigender Weise nicht möglich ist. Zwar existieren verschiedenste Lernverfahren zur Konvergenzbeschleunigung, wie etwa RProp [79], doch finden sich keine Verfahren, die neue Informationen verlustfrei im On-Line-Sinne nachtrainieren können. Verwendet man klassische Lernverfahren wie etwa Backpropagation [83], so wird schnell klar, daß Nachtrainieren neuer Information zu unvorhersehbarem Vergessen bereits gelernter Information führt. Der Kern des Problems liegt hier in der Art und Weise der Informationsspeicherung neuronaler Netze. Diese findet nicht lokal sondern global statt, so daß die Modifikation eines Gewichtes Auswirkungen auf *alle bisher gelernten Informationen* hat.

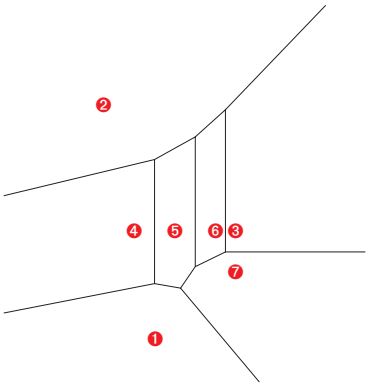


Abbildung 5.1: Beispiel einer Voronoi-Parkettierung von Clusterzentren

Will man dies verhindern, so muß zu einer *lokalen* Informationsspeicherung übergegangen werden, die *lokales* Lernen ermöglicht. Hierzu wurde ein Ansatz entwickelt, der einen beliebigen Prozeßraum On-Line in verschiedene Teilräume aufteilt und diesen jeweils eigenständige lokale Modelle zuweist. Neben dem besser definierten Lernverhalten gibt es noch weitere Gründe

diese Vorgehensweise zu wählen. Zum einen können so Unstetigkeiten modelliert werden, zum anderen ist eine völlig freie Modellwahl möglich. So kann an einer Stelle ein lineares Modell, an anderer ein Multi-Layer-Perzeptron oder ähnliches genutzt werden, was die Nachbildung verschiedenster Dynamiken erlaubt. In [96] wurde diese Aufteilung noch anhand von Regeln erzielt, mit dem Nachteil, daß eine genaue Kenntnis des Prozesses notwendig ist. Aus diesem Grund wird die Aufteilung unter Zuhilfenahme der Vorhersagegüte der jeweiligen lokalen Modelle automatisch durchgeführt, was in Abschnitt 5.2 näher erläutert ist.

Um einen beliebigen Prozeßraum in verschiedene Teilräume aufzuteilen, können unterschiedliche Techniken angewandt werden. So ist zum Beispiel eine Voronoi-Parkettierung – wie sie in Abbildung 5.1 gezeigt wird – mit Clusterzentren in den Datenschwerpunkten[1] und einer euklidischen Metrik üblich. Untersucht man diese Art der Voronoi-Parkettierung unter den gegebenen Randbedingungen, wie etwa Erfassung von Sonderfällen innerhalb eines Teilraumes, Anpassung eines Teilraumes an langsame Veränderungen usw., so stellt man zwangsläufig fest, daß die Erfassung von Teilräumen durch Ellipsoide die wesentlich flexiblere Lösung ist. Die folgenden Vorteile sind unmittelbar einsichtig:

- Einfache Teilraumbeschreibung durch Angabe eines Ellipsoidzentrum $\vec{c}$ und einer zugeordneten positiv definiten Matrix $\mathbf{E}$, die die Ausdehnung des Ellipsoids angibt.
- Einfache Anpassung an die Änderung der Teilraumgeometrie durch Verzerrung eines vorhandenen Ellipsoids. Dies entspricht einer einfachen Matrixmultiplikation. Die Modellierung von Sonderfällen kann durch Ellipsoide innerhalb von Ellipsoiden abgedeckt werden.
- Vielfältige Visualisierungsmöglichkeiten der entstehenden Strukturen durch Projektion der Ellipsoidzentren und zugehörigen Matrix auf zwei oder drei Dimensionen.

Obige Überlegungen haben dazu geführt, eine lokale Modellbildungsmethodik auf der Basis von ellipsoiden Karten zu entwickeln. Im folgenden werden zuerst einige Grundlagen zu ellipsoiden Karten und im Anschluß daran die Kopplung mit der lokalen Modellbildungskomponente dargestellt.

5.1 Ellipsoide Karten

Ein beliebiges Hyperellipsoid kann durch Angabe seines Schwerpunktes und einer positiv definiten Matrix vollständig charakterisiert werden.

Im folgenden werden einige grundlegend Definitionen eingeführt, die für die weitere Betrachtung notwendig sind. Darauf aufbauend wird anschließend eine spezielle Metrik entworfen, die es erlaubt, bekannte Clusterverfahren (wie etwa „k-Nearest-Neighbour“ beschrieben unter anderem von Dasarathy [18] oder „Learning-Vector-Quantisation“ von Kohonen [53]) zu verwenden, um Raumregionen über Clusterzugehörigkeiten zu definieren.

5.1.1 Definition eines Ellipsoids

Allgemein läßt sich ein Hyperellipsoid beschreiben durch die Quadrik:

[1] Die Zahl der Zentren wird in der Regel vorgegeben und anschließend werden diese durch ein unüberwachtes Lernverfahren wie etwa LVQ [53] positioniert.

$$(\mathbf{E}\vec{x})^T \underbrace{(\mathbf{E}\vec{x})}_{\vec{x}_e} = 1 \tag{5.1}$$

Die Matrix E ist positiv definit und hat die Form:

$$\mathbf{E} = \mathbf{Q}_1\mathbf{D}\mathbf{Q}_2 \quad \text{mit} \quad \mathbf{Q}_1, \mathbf{Q}_2 \in \mathcal{O}_{\text{Rotation}},\ \mathbf{D} = \begin{pmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_n \end{pmatrix} \tag{5.2}$$

Man führt also eine Hyperkugel mit Radius 1 durch die Drehung $\mathbf{Q}_1^{-1}$, die achsparallele lineare Streckung $\mathbf{D}^{-1}$, und die Drehung $\mathbf{Q}_2^{-1}$ in ein Hyperellipsoid über. In Abbildung 5.2 wird die Vorgehensweise anhand eines 2-dimensionalen Beispiels demonstriert:

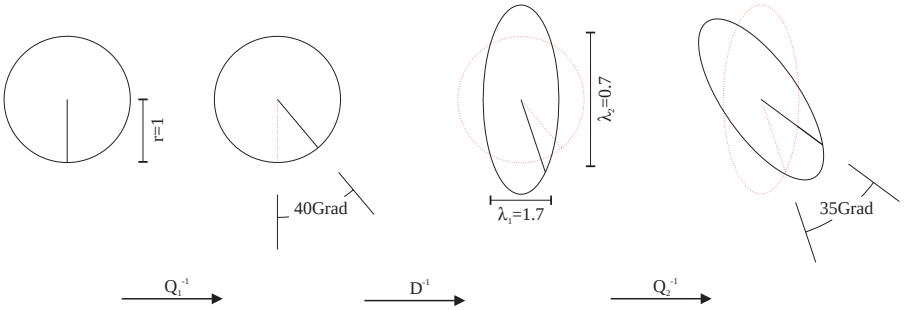


Abbildung 5.2: Überführen eines Kreises in ein Ellipsoid

Da alle Rotationsmatrizen orthonormal sind und somit $\mathbf{Q}_1^T = \mathbf{Q}_1^{-1}$ gilt, läßt sich Formel (5.1) und (5.2) überführen in:

$$\vec{x}^T(\mathbf{Q}_2^T\mathbf{D}^2\mathbf{Q}_2)\vec{x} = (\mathbf{Q}_2\vec{x})^T\mathbf{D}^2\underbrace{(\mathbf{Q}_2\vec{x})}_{\vec{x}_r} = 1 \tag{5.3}$$

Offensichtlich existieren also verschiedene Matrizen $\mathbf{E}$, die dasselbe Hyperellipsoid definieren. Es ist leicht einzusehen, daß die erste Rotation $\mathbf{Q}_1$ tatsächlich beliebig sein kann, da sie im System der Hyperkugel stattfindet, die durch eine Rotation immer auf sich selbst abgebildet wird. Im folgenden wird deshalb $\mathbf{Q}_1$ vernachlässigt und von der normalisierten *eindeutigen* Form $\mathbf{E} = \mathbf{DQ}$ ausgegangen. Die Größe eines Ellipsoids läßt sich am besten über sein Volumen charakterisieren, das folgendermaßen erhalten werden kann:

$$V_{\mathbf{E}} = \underbrace{V_{\text{Kugel}}}_{=1} \cdot \det(\mathbf{E}^{-1}) = \det(\mathbf{D}^{-1})\underbrace{\det(\mathbf{Q}^T)}_{=1} = \prod_{i=1}^{n} \lambda_i^{-1} \tag{5.4}$$

5.1.2 Definition der Zugehörigkeit zu einem Ellipsoid

Sei eine Menge von m Hyperellipsoiden mit je einer zugeordneten Abstandsfunktion $\mathfrak{D}$ gegeben durch $\{\{\vec{c}_1, \mathbf{E}_1, \mathfrak{D}_1\}, \{\vec{c}_2, \mathbf{E}_2, \mathfrak{D}_2\}, \ldots \{\vec{c}_m, \mathbf{E}_m, \mathfrak{D}_m\}\}$. Die Zugehörigkeit eines Punktes $\vec{p}$ zu einem Hyperellipsoid $\{\vec{c}_k, \mathbf{E}_k\}$ wird über die jeweilige Abstandsfunktion $\mathfrak{D}_i : \mathbb{R}^n \to \mathbb{R}^+$ definiert und im folgenden mit dem Zeichen $\in$ ausgedrückt. Es gilt:

$$\begin{aligned} \vec{p} &\in \{\vec{c}_k, \mathbf{E}_k\} \\ &\Leftrightarrow \\ 1 \leq i < k \Rightarrow \mathfrak{D}_i(\vec{p}) > \mathfrak{D}_k(\vec{p}) \quad &\wedge \quad k < i \leq m \Rightarrow \mathfrak{D}_i(\vec{p}) \geq \mathfrak{D}_k(\vec{p}) \end{aligned} \tag{5.5}$$

Hierbei werden Punkte, die von verschiedenen Clusterzentren denselben Abstand haben, dem ersten Cluster zugeordnet. Dadurch wird eine feste Sortierung der Hyperellipsoide vorausgesetzt.

Ignoriert man die ellipsoide Matrix $\mathbf{E}_i$ und verwendet nur den euklidischen Abstand zum Schwerpunkt $\vec{c}_i$, d.h. $\mathfrak{D}_i : \vec{p} \mapsto \|\vec{p} - \vec{c}_i\|$, so erhält man die bereits in Abbildung 5.1 gezeigte Voronoi-Parkettierung als Zugehörigkeitsstruktur.

Um auch die ellipsoide Struktur der einzelnen Cluster zu berücksichtigen, wird im weiteren das in [49] beschriebene heuristische Abstandsmaß schrittweise modifiziert, bis die folgenden Randbedingungen erfüllt sind:

1. Befindet sich ein Punkt innerhalb genau eines Ellipsoids, so gehört er diesem an.
2. Befindet sich ein Punkt innerhalb mehrerer Ellipsoide, so gehört er dem kleinsten an.
3. Befindet sich ein Punkt außerhalb aller Ellipsoide, so gehört er dem nächsten im Bezug auf den Abstand zur Oberfläche an.

5.1.3 Das heuristische Abstandsmaß

In [49] wurden ellipsoide Karten verwendet, um ein Freiraummodell im Konfigurationsraum eines Roboterarms zu erhalten. Bedingt durch diese Aufgabenstellung ist es hierbei nur wichtig, Raum innerhalb und Raum außerhalb der Ellipsoide unterscheiden zu können und zur Adaption für außerhalb liegende Punkte festzulegen, welches Ellipsoid das nächstliegendste ist. Das dort verwendete Abstandsmaß lautet

$$\mathfrak{D}_i(\vec{p}) := \begin{cases} \|\mathbf{E}_i(\vec{p} - \vec{c}_i)\| & \text{innerhalb } (\equiv \|\mathbf{E}_i(\vec{p} - \vec{c}_i)\| \leq 1) \\ \|\vec{p} - \vec{c}_i\|(1 - \frac{1}{\|\mathbf{E}_i(\vec{p} - \vec{c}_i)\|}) + 1 & \text{außerhalb} \end{cases} \tag{5.6}$$

Wie in Abbildung 5.3 veranschaulicht, hat die Ellipsoidoberfläche immer den Abstand 1 zum Schwerpunkt $\vec{c}_i$. Der Abstand eines Punktes innerhalb des Ellipsoids wird im System der Hyperkugel mit Radius Eins bestimmt. Das Lot eines Punktes außerhalb des Ellipsoids auf die Ellipsoidoberfläche wird durch den Schnittpunkt dieser mit der Verbindungslinie zum Schwerpunkt genähert, wie es in Abbildung 5.3 gezeigt ist. Durch Addition von Eins erhält man den genäherten Außenabstand.

Diese Heuristik birgt im Hinblick auf die in Abschnitt 5.1.2 definierten Randbedingungen folgende Schwächen:

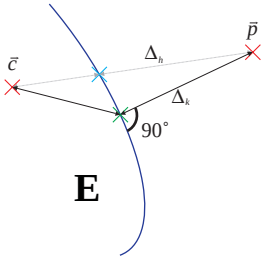


Abbildung 5.3: Näherung des Lotabstandes Δ_k auf die Ellipsoidoberfläche durch Δ_h, den Abstand zum Schnittpunkt der Verbindungslinie zum Zentrum.

1. Der Abstand der Ellipsoidoberfläche vom Schwerpunkt ist unabhängig von der Ellipsoidgröße. Dies hat zur Folge, daß kleine Ellipsoide, die innerhalb eines größeren Ellipsoids liegen, Teile ihres Einzugsgebietes verlieren. Dieser Effekt ist von der Entfernung der beiden Ellipsoidschwerpunkte zueinander abhängig.

2. In Fällen stark verzerrter Ellipsoide führt die Verwendung obiger Näherung des Außenabstandes zur Ellipsoidoberfläche zu so starken Abweichungen von der korrekten Lösung, daß falsche Modellzuordnungen unvermeidlich sind.

Die beschriebenen Schwächen werden in Abbildung 5.7a auf Seite 62 an einem Beispiel verdeutlicht. Zur Beseitigung des Problems (1) wird das nun folgende Abstandsmaß eingeführt, welches anschließend erweitert wird um auch Problem (2) zu beheben.

5.1.4 Das modifizierte heuristische Abstandsmaß

Zuerst wird durch Subtraktion von Eins der Nullpunkt des heuristischen Abstandmaßes auf die Ellipsoidoberfläche verschoben. Abstände von Punkten innerhalb eines Ellipsoids sind damit immer kleiner als Abstände von Punkten außerhalb.

Im zweiten Schritt werden die negativen Abstände innerhalb eines Ellipsoids zusätzlich mit dem inversen Radius einer Hyperkugel mit Volumen $V_{\mathbf{E}_i}$ skaliert. Befindet sich ein Punkt innerhalb mehrerer Ellipsoide mit unterschiedlichem Volumen, so liegt der Punkt tendenziell dem Zentrum des kleinsten Ellipsoids am nächsten.

$$\mathfrak{D}_i(\vec{p}) := \begin{cases} V_{\mathbf{E}_i}^{-\frac{k}{n}}(\|\mathbf{E}_i(\vec{p}-\vec{c}_i)\|-1) & \text{innerhalb } (\equiv \|\mathbf{E}_i(\vec{p}-\vec{c}_i)\| \leq 1) \\ \|\vec{p}-\vec{c}_i\|(1-\frac{1}{\|\mathbf{E}_i(\vec{p}-\vec{c}_i)\|}) & \text{außerhalb} \end{cases} \tag{5.7}$$

Der Parameter k gibt dabei an, wie stark die Skalierung des Abstandes innerhalb eines Ellipsoides vom mittleren Ellipsoidradius abhängt.

Um zu veranschaulichen, wie sich die Wahl von k auf das Abstandsmaß auswirkt, begibt man sich am besten zurück in das Kreissystem, betrachtet allerdings nicht den Einheitskreis, sondern zwei konzentrische Kreise mit den Zentren $\vec{c}_1 = \vec{c}_2$ im Ursprung und unterschiedlichen Radien r_1 und r_2. Befindet sich ein Punkt $\vec{p}$ innerhalb beider Kreise, so gilt nach (5.7) für seinen Abstand d_i von der $i$ten Kreisoberfläche:

$$d_i : \vec{p} \mapsto \frac{1}{r_i^k} \frac{\|\vec{p} - \vec{c_i}\| - r_i}{r_i} = \frac{\|\vec{p}\| - r_i}{r_i^{k+1}}$$

Abbildung 5.4 zeigt den Verlauf obiger Abstände in Abhängigkeit von $\|\vec{p}\|$ an einem Beispiel. Man erkennt deutlich die Wirkung des Faktors $\frac{1}{r_i^k}$ auf die Steigung der abgebildeten Geraden, deren Schnittpunkt genau die Voronoi-Grenze b darstellt.

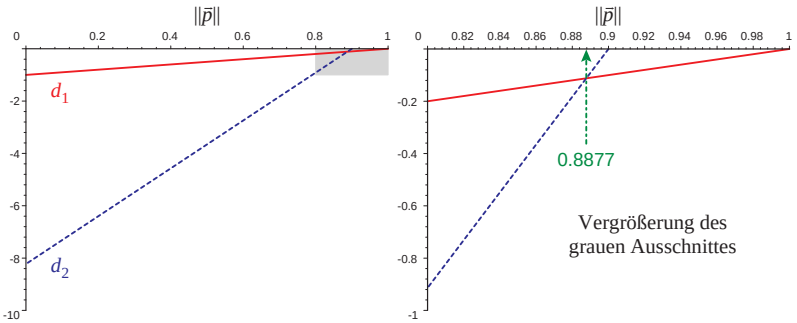


Abbildung 5.4: Der Verlauf von d_1 und d_2 für $r_1 = 1$, $r_2 = 0.9$ und $k = 20$. Man erhält aus Formel (5.8) den theoretischen Schnittpunkt der beiden Geraden, und damit die Voronoi-Grenze b zu 0.8877. Im vergrößerten Ausschnitt ist zu erkennen, daß dieser mit dem grünen Pfeil markierte Wert sehr nahe am optimalen Wert von 0.9, dem Rand des inneren Kreises, liegt.

Setzt man $d_1(\vec{p}) = d_2(\vec{p})$ und löst dies nach $\|\vec{p}\|$ auf, so ergibt sich die Voronoi-Grenze b zwischen den beiden Kreisen in Abhängigkeit von r_1 und r_2:

$$b : (r_1, r_2) \mapsto \frac{r_1 r_2^{k+1} - r_2 r_1^{k+1}}{r_2^{k+1} - r_1^{k+1}} \tag{5.8}$$

Desweiteren läßt sich sofort ableiten, daß $\lim_{r_1 \to r_2} b(r_1, r_2) = r_2 \frac{k}{k+1}$. Die Voronoi-Grenze wandert erst für $k \to \infty$ auf die Ellipsoidoberfläche. Es ist wünschenswert, k möglichst groß zu wählen. Hierbei sollte berücksichtigt werden, daß für zu große k numerische Probleme auftreten können. In der Praxis hat sich ein Wert von 20 als sinnvolle Alternative bewährt. Abbildung 5.5 zeigt den Verlauf der Voronoi-Grenze für verschiedene k.

Offensichtlich sollte die optimale Voronoi-Grenze entlang der Identität $b = r_1$ bis $r_1 = r_2$ und dann entlang der Geraden $b = r_2$ verlaufen. Wie man erkennen kann, erfüllt die Kurve für $k = 1$ diese Bedingung nicht im geringsten, für $k = 20$ jedoch relativ gut.

Nach dieser Modifikation des Innenabstandes wird im folgenden noch die Heuristik zur Schätzung des Außenabstandes durch das tatsächliche Lot auf die Ellipsoidoberfläche ersetzt.

5.1.5 Das korrekte Abstandsmaß

Gesucht ist der minimale Abstand eines beliebigen Vektors $\vec{p}$ zur Oberfläche des Hyperellipsoids $\mathbf{E}$. Es ist also die Länge des Vektors $\vec{d} = \vec{x} - \vec{p}$ zu minimieren, wobei $\vec{x}$ ein beliebiger Punkt

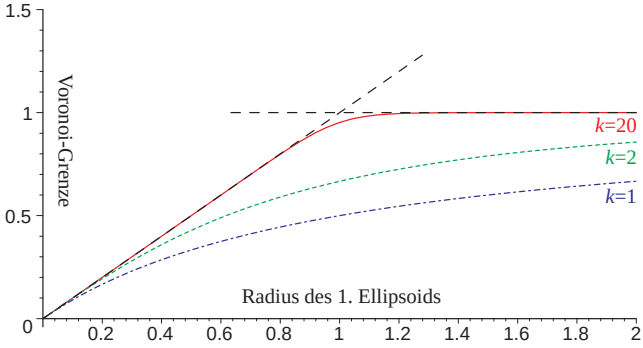


Abbildung 5.5: Plot der Voronoi-Grenze b zweier Kreise in Abhängigkeit von Radius r_1 bei fixem Radius $r_2 = 1$. Die gestrichelte Linie markiert die Identität $b = r_1$ und die Gerade $b = r_2$.

der Ellipsoidoberfläche ist. Da die Länge des Vektors $\vec{d}$ rotationsinvariant ist, betrachtet man zur Vereinfachung alternativ den Vektor:

$$\vec{d}_r = \mathbf{Q}(\vec{x} - \vec{p}) = \vec{x}_r - \underbrace{\mathbf{Q}\vec{p}}_{\vec{p}_r} \tag{5.9}$$

Zur Lösung dieses Optimierungsproblems muß eine parametrierte Darstellung des Abstandsvektors $\vec{d}_r$ gefunden werden. Diese erhält man am einfachsten aus einer Winkelparametrierung der Hyperkugel mit Radius Eins, wie etwa:

$$\vec{x}_e : \vec{\alpha} \mapsto \begin{pmatrix} \cos(\alpha_1) \prod_{i=2}^{n-1} \cos(\alpha_i) \\ \sin(\alpha_1) \prod_{i=2}^{n-1} \cos(\alpha_i) \\ \vdots \\ \sin(\alpha_{n-2}) \cos(\alpha_{n-1}) \\ \sin(\alpha_{n-1}) \end{pmatrix} \tag{5.10}$$

Unter Ausnutzung von $\mathbf{D}\vec{x}_r = \vec{x}_e$ erhält man aus (5.10) und (5.9) die gesuchte Parametrierung des Abstandvektors $\vec{d}_r$. Ersetzt man λ_i^{-1} durch η_i, die Länge der $i$ten Halbachse des Ellipsoids, so lautet das Ergebnis:

$$\vec{d}_r : \vec{\alpha} \mapsto \begin{pmatrix} \eta_1 \cos(\alpha_1) \prod_{i=2}^{n-1} \cos(\alpha_i) - p_{r,1} \\ \eta_2 \sin(\alpha_1) \prod_{i=2}^{n-1} \cos(\alpha_i) - p_{r,2} \\ \vdots \\ \eta_{n-1} \sin(\alpha_{n-2}) \cos(\alpha_{n-1}) - p_{r,n-1} \\ \eta_n \sin(\alpha_{n-1}) - p_{r,n} \end{pmatrix} \tag{5.11}$$

Die Minimierung von $\|\vec{d}_r\|$ führt zu Δ_k aus Abbildung 5.3 und kann auf zwei verschiedene Weisen angegangen werden. Einerseit wäre eine analytische Lösung denkbar, die für beliebige $\{\vec{c}_i, \mathbf{E}_i\}$ und $\vec{p}$ die exakte Länge des Lotes angibt. Andererseits kann mit Hilfe der Jacobi-Matrix ein numerisches Optimierungsverfahrens, wie etwa der von Moré [65] beschriebene Levenberg-Marquardt-Algorithmus, angewandt werden. Da die Oberfläche eines Ellipsoids

konvex ist, existiert nur *eine*, *eindeutige* Lösung ohne lokale Minima. Somit kann mit Kenntnis der Jacobi-Matrix bereits der erste Optimierungsschritt in Richtung des globalen Minimums ausgeführt und nach wenigen Schritten eine bis auf Maschinengenauigkeit ε korrekte Lösung erzielt werden. Die benötigte Jacobi-Matrix erhält man aus (5.11) durch partielles Ableiten nach $\vec{\alpha}$. Wird $\sin(\alpha_i)$ durch s_i und $\cos(\alpha_i)$ durch c_i ersetzt, so lautet die gesuchte Jacobi-Matrix:

$$\begin{pmatrix}
-\eta_1 s_1 \prod_{k=2}^{n-1} c_k & -\eta_1 c_1 s_2 \prod_{k=2}^{n-1} c_k & & \dots & -\eta_1 s_j \prod_{k=1}^{j-1} c_k \prod_{k=j+1}^{n-1} c_k & \dots & -\eta_1 s_{n-1} \prod_{k=1}^{n-2} c_k \\
\eta_2 \prod_{k=1}^{n-1} c_k & -\eta_2 s_1 s_2 \prod_{k=3}^{n-1} c_k & & \dots & -\eta_2 s_1 s_j \prod_{k=2}^{j-1} c_k \prod_{k=j+1}^{n-1} c_k & \dots & -\eta_2 s_1 s_{n-1} \prod_{k=2}^{n-2} c_k \\
0 & \eta_3 \prod_{k=2}^{n-1} c_k & -\eta_3 s_2 s_3 \prod_{k=4}^{n-1} c_k & \dots & -\eta_3 s_2 s_j \prod_{k=3}^{j-1} c_k \prod_{k=j+1}^{n-1} c_k & \dots & -\eta_3 s_2 s_{n-1} \prod_{k=3}^{n-2} c_k \\
\vdots & 0 & \eta_4 \prod_{k=3}^{n-1} c_k & \dots & -\eta_4 s_3 s_j \prod_{k=4}^{j-1} c_k \prod_{k=j+1}^{n-1} c_k & \dots & -\eta_4 s_3 s_{n-1} \prod_{k=4}^{n-2} c_k \\
\vdots & \vdots & 0 & & \vdots & & \vdots \\
\vdots & \vdots & \vdots & & & & \\
0 & 0 & 0 & \dots & 0 & 0 & -\eta_n c_{n-1}
\end{pmatrix}$$

Im folgenden sollen die zwei verschiedenen Lösungsansätze vorgestellt und diskutiert werden.

Analytische Lösung

Eine notwendige Bedingung, die erfüllt sein muß, wenn $\|\vec{d}_r\|$ minimal ist lautet:

$$2\vec{d}_r^T \frac{\partial \vec{d}_r}{\partial \vec{\alpha}} = \vec{0} \tag{5.12}$$

Offensichtlich existieren jeweils bis zu $2n$ reelle Lösungen, da es pro Halbachse zwei Extrema geben kann. Somit muß der minimale Abstand aus der Lösungsmenge für (5.12) nochmals extra ausgewählt werden.

Betrachtet man als *minimales* Beispiel den Fall $n = 2$, so erhält man aus (5.12) unter Ausnutzung von $\cos(\alpha_1) = \sqrt{1 - \sin(\alpha_1)^2}$ und Substitution von s für $\sin(\alpha_1)$ folgende Gleichung:

$$-2\eta_1^2 s\sqrt{1-s^2} + 2\eta_1 p_1 s - 2\eta_2 p_2 \sqrt{1-s^2} + 2\eta_2^2 s\sqrt{1-s^2} = 0 \tag{5.13}$$

Offensichtlich müssen zur Lösung dieses einfachen Falles bereits die Nullstellen eines Polynoms 4. Ordnung gefunden werden. Im allgemeinen Fall muß ein System mit $n-1$ Gleichungen multivariater Polynome gelöst werden, um die Winkel $\alpha_1, \dots, \alpha_{n-1}$ zu erhalten.

Zur Lösung dieses aus (5.12) resultierenden Gleichungssystems können z.B. Term-Rewriting-Verfahren wie etwa der Buchberger-Algorithmus [13, 12] genutzt werden. Diese Verfahren sind jedoch exponentiell in der Zahl der Variablen. An dieser Stelle wird schon klar, daß eine analytische Lösung zwar möglich, aber zu komplex ist, um den Mehraufwand gegenüber einer numerischen Lösung zu rechtfertigen. Aus diesem Grund wurde die im folgenden beschriebene numerische Variante gewählt.

Numerische Näherung

Wie bereits erwähnt, kann mit Hilfe eines Optimierers und der Jacobi-Matrix die Länge des Vektors $\vec{d}_r$ minimiert werden. Als Startwert sollte eine Winkelparametrierung vorgegeben werden, die möglichst nah am zu findenden Minimum liegt. Es bietet sich an, den in Abbildung 5.3

gezeigten Schnittpunkt zu wählen, also mit der alten Heuristik Δ_h als Startwert zu beginnen. Setzt man $\vec{p}_e = \mathbf{E}\vec{p}$, so führt dies zur folgenden Belegung von $\vec{\alpha}$:

$$\alpha_i = \begin{cases} \sin^{-1}\left(\frac{p_{e,i+1}}{\|\vec{p}_e\| \cdot \prod_{k=i+1}^{n-1} \cos(\alpha_k)}\right) & \text{wenn } i \neq 1 \\ \cos^{-1}\left(\frac{p_{e,1}}{\|\vec{p}_e\| \cdot \prod_{k=2}^{n-1} \cos(\alpha_k)}\right) \cdot \operatorname{sgn}(p_{e,2}) & \text{sonst} \end{cases} \tag{5.14}$$

Zu beachten ist dabei der Sonderfall $\exists k \in \{1 \cdots n-1\} : p_{e,k+1} = \|\vec{p}_e\|$, denn daraus folgt $\cos(\alpha_k) = 0$, wodurch Formel (5.14) ungültig wird. Es gilt dann jedoch offensichtlich:

$$\alpha_i = \begin{cases} 0 & \text{wenn } i \neq k \\ \frac{\pi}{2} & \text{sonst} \end{cases} \tag{5.15}$$

Abbildung 5.6 zeigt die Wirkung der Optimierung an einem 2-dimensionalen Beispiel. Die verschiedenen Voronoi-Regionen sind unterschiedlich farbig dargestellt. Man erkennt deutlich, daß im Mittel bereits der erste Optimierungsschritt das Minimum erreicht. Der verwendete Levenberg-Marquardt-Optimierer aus der MINPACK-1 Bibliothek [66] verfügt über eine automatische Schrittweitenanpassung, die zu dem Effekt im grau unterlegten Bereich führt. Dort befindet sich das gesuchte Minimum links unten im roten Bereich des Ellipsoids mit der Nummer eins, also im Bereich der maximalen Krümmung. Zur exakten Bestimmung des Lotes muß die Schrittweite an dieser heruntergesetzt werden, und die Zahl der nötigen Schritte steigt.

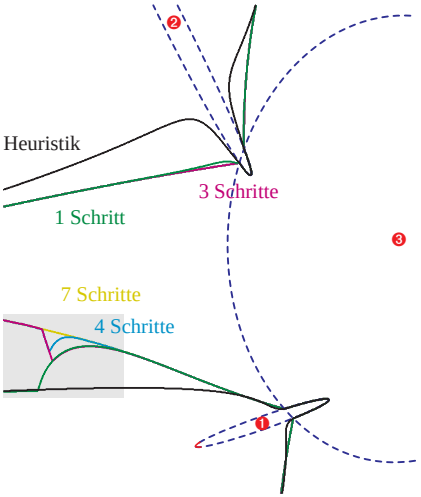


Abbildung 5.6: Voronoi-Parkettierung in Abhängigkeit von der Zahl der Optimierungsschritte

5.1.6 Komplexitätsreduktion des korrekten Abstandsmaßes

Um ein Gefühl für die Komplexität des Optimierungsproblems zu erlangen, wurde eine Statistik über die Auswertungsdauer und Zahl der nötigen Optimiererschritte zur Berechnung von

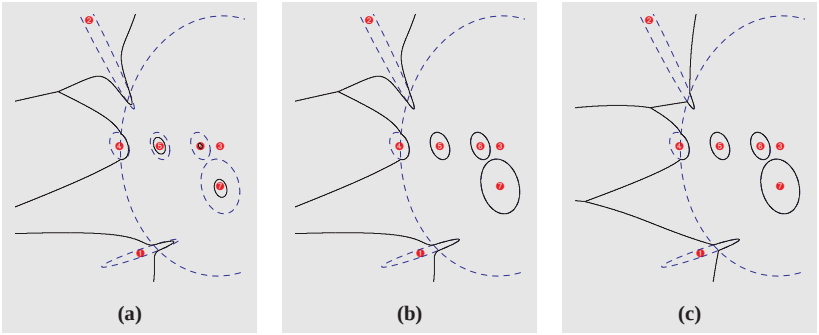


Abbildung 5.7: Vergleich der Voronoi-Parkettierung bei Verwendung verschiedener Metriken. **(a)** Nutzung der Heuristik für Außenabstände und unskalierter Innenabstände führt zu Zuordnungsfehlern inner- und außerhalb der Ellipsoide. **(b)** Nutzung der Heuristik für Außenabstände und volumenskalierter Innenabstände korrigiert Zuordnungsfehlern innerhalb der Ellipsoide. **(c)** Nutzung der korrekten Außenabstände und volumenskalierter Innenabstände führt zu korrekten Zuordnungen inner- und außerhalb der Ellipsoide.

Δ_k bei zufällig generierten Ellipsoiden $\{\vec{c}_i, \mathbf{E}_i\}$ und zufällig gewählten Punkten $\vec{p}$ für Eingabedimensionen von 2 bis 100 mit je 10000 Stichproben erstellt. Abbildung 5.8 zeigt eine Grafik, die aus dieser Statistik erstellt wurde. Man erkennt deutlich das exponentielle Ansteigen der Auswertungsdauer bzw. das logarithmische Ansteigen der Zahl der benötigten Optimiererschritte. Da Abbildung 5.6 in einigen Bereichen kaum, in anderen hingegen sehr starke Abweichungen zwischen der Heuristik und der korrekten Variante zeigt, wurde auch der relative Fehler $\frac{\Delta_h - \Delta_k}{\Delta_k}$ erfaßt. Der mittlere relative Fehler lag bei 5.06%, der maximale relative Fehler betrug sogar 172%.

Die Verwendung des korrekten Abstandsmaßes ist also einerseits notwendig, andererseits aber durch den notwendigen Optimierungslauf eine relativ teure Angelegenheit, speziell wenn man höherdimensionale Räume betrachtet. Bei der Suche des nächsten Nachbarn ist aber die Kenntnis des exakten Abstandes nur zum momentan nächsten Clusterzentrum notwendig, da zur Ablehnung eines weiteren Clusterzentrums bereits die Aussage „*Abstand ist größer*“ ausreicht. Besitzt man eine Abschätzung des maximalen Fehlers $\Gamma_{\max}$, für die $\Delta_k \geq \Delta_h - \Gamma_{\max}$ immer gilt, so kann man zuerst testen, ob nicht bereits $\Delta_h - \Gamma_{\max}$ größer als der momentane Minimalabstand ist, und den korrekten Abstand nur bei Bedarf berechnen.

Zur Abschätzung des maximal möglichen Fehlers betrachten wir nochmals Formel (5.13). Vernachlässigt man für $\eta_1 \ll \eta_2$ alle η_1-Terme, so vereinfacht sich das Polynom und die Nullstelle $\frac{p_2}{\eta_2}$ kann als Lösung des Minimierungsproblems gefunden werden. Damit erhält man als Schätzung für den korrekten Abstand des Punktes $\vec{p}$ zur Ellipsoidoberfläche:

$$\left| p_1 - \eta_1 \sqrt{1 - \frac{p_2^2}{\eta_2^2}} \right| \overset{\eta_1 \ll \eta_2}{\longrightarrow} |p_1 - \eta_1| \tag{5.16}$$

Zu beachten ist, daß für $p_2 \to \eta_2$ oder $\eta_1 \to 0$ obiger Term gegen p_1 konvergiert. Dennoch

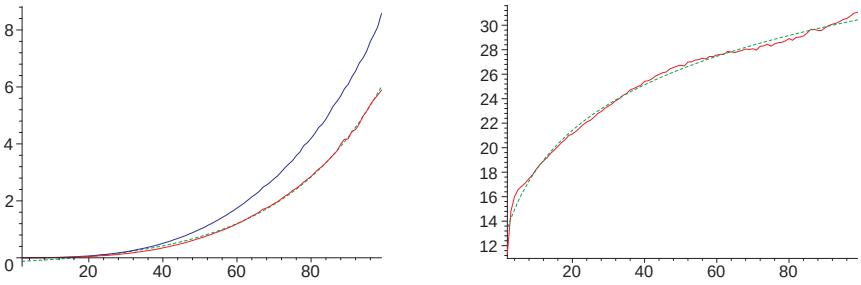


Abbildung 5.8: Statistik über die Auswertungsdauer und Zahl der nötigen Optimiererschritte zur optimierten Abstandsberechnung auf einer SUN Ultra-60 Workstation mit zwei 296 MHz Prozessoren. Die linke Grafik ist der Auswertungsdauer, die rechte der Optimiererschrittzahl zugeordnet. Die rote Kurve zeigt jeweils die statistischen Ergebnisse, die grün gestrichelte Kurve hingegen eine exponentielle bzw. logarithmische Tendenz, die an die gemessenen Punkte mittels Regression angepaßt wurde. Zusätzlich wurde die Auswertungsdauer auf einer SUN Ultra-2 Workstation mit zwei 200 MHz Prozessoren erfaßt, die um den Faktor $\frac{2}{3}$ langsamer ist und zum Vergleich als blaue Kurve eingezeichnet.

bleibt der lineare Zusammenhang zu p_1 erhalten. Durch Subtraktion der Formel (5.16) von der Abstandheuristik aus Formel (5.7) ergibt sich eine Schätzung für den absoluten Fehler:

$$\sqrt{p_1^2+p_2^2}\left(1-\frac{1}{\sqrt{\eta_1^{-2}p_1^2+\eta_2^{-2}p_2^2}}\right)-\left|p_1-\eta_1\sqrt{1-\frac{p_2^2}{\eta_2^2}}\right| \tag{5.17}$$

Wird nur der positive Quadrant betrachtet, und wird der Punkt $\vec{p}$ durch Einsetzen von $p_1=\sin(\alpha)$, $p_2=\sqrt{1-\sin(\alpha)^2}$ für $\alpha\in[0,\frac{\pi}{2}]$ auf einem Kreis mit Radius 1 um den Ursprung bewegt, so vereinfacht sich der Ausdruck für $\eta_2\equiv 1$ zu:

$$\Gamma(\alpha)=1+\sin(\alpha)(\eta_1-1)-\frac{1}{\sqrt{1+\sin(\alpha)^2(\eta_1^{-2}-1)}} \tag{5.18}$$

Gesucht ist $\alpha_{\max}(\eta_1)$, das obige Gleichung maximiert. Durch Ableiten nach α und Nullsetzen der resultierenden Formel erhält man eine Gleichung mit mehreren Lösungen. Mit Hilfe einer Fallunterscheidung läßt sich schließlich das gesuchte $\alpha_{\max}(\eta_1)$ finden. Der zugehörige maximale Fehler $\Gamma_{\max}$ ergibt sich durch Einsetzen von $\alpha_{\max}$ in die Funktion $\Gamma(\alpha)$. Abbildung 5.9 zeigt den maximalen Fehler und $\alpha_{\max}$ für $\eta_1\in[0,0.1]$.

Mit zunehmender Verzerrung des Ellipsoids ($\eta_1\ll\eta_2$) nimmt der maximale Fehler $\Gamma_{\max}$ ebenfalls zu. Der Grenzwert von $\Gamma_{\max}$ für $\eta_1\to 0$ ist in diesem Fall Eins. Man kann sich leicht veranschaulichen, daß im allgemeinen Fall $\lim_{\eta_1\to 0}\Gamma_{\max}=\eta_2$ gilt.

Um eine sinnvolle Fehlerabschätzung durchführen zu können, wurde $\Gamma_{\max}$ für verschiedene $\eta_1,\ \eta_2\in]0,1]$ und $\|\vec{p}\|\equiv 1$, $\mathbf{Q}\equiv\mathbf{I}$ numerisch mit Hilfe eines Bisektionsverfahrens bestimmt. Die Werte für $\eta_1\equiv 0$ bzw. $\eta_2\equiv 0$ können hierbei numerisch nicht bestimmt, jedoch durch die zugehörigen Grenzwerte η_2 bzw. η_1 ersetzt werden.

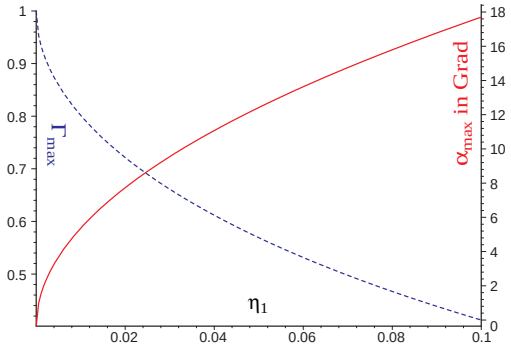


Abbildung 5.9: Maximaler Fehler $\Gamma_{\max}$ und Winkel $\alpha_{\max}$ für $\eta_2 \equiv 1$.

Mit Hilfe der so gefundenen Fehlerabschätzung für 2-dimensionale Ellipsoide kann auch für beliebige Dimensionen eine Fehlerabschätzung durchgeführt werden. Hierzu ist lediglich die Berechnung des größten und kleinsten η notwendig, welche die maximal mögliche Verzerrung des Ellipsoids bestimmen. Verwendet man $\eta_{\min}$ und $\eta_{\max}$ anstelle von η_1 und η_2, so erhält man $\Gamma_{\max}$ für den n-dimensionalen Fall. Das Ergebnis der Berechnung von $\Gamma_{\max}$ bzw. $\alpha_{\max}$ ist in Abbildung 5.10 dargestellt.

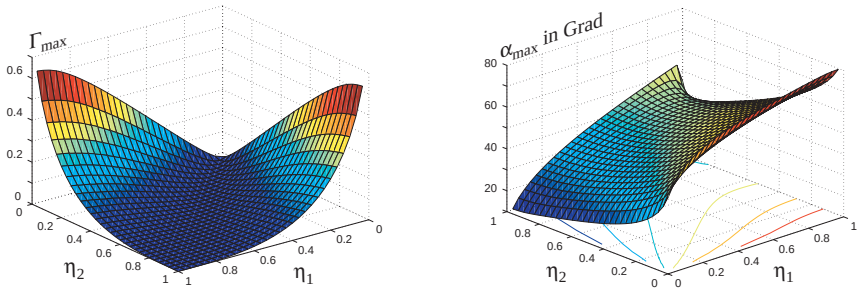


Abbildung 5.10: Die linke Fläche stellt den maximalen Fehler $\Gamma_{\max}$ bei Verwendung der Heuristik aus Formel (5.7) anstelle der korrekten Variante aus Formel (5.12) an einem 2-dimensionalen Beispiel für $\eta_1,\ \eta_2 \in]0,1]$ dar. Für $\vec{p} = (\sin(\alpha)\ \cos(\alpha))^T$ wurde jeweils der maximale Fehler mit Hilfe eines Bisektionsverfahrens durch Variation von $\alpha \in [0, \frac{\pi}{2}]$ bestimmt. Die rechte Fläche zeigt den zugehörigen Winkel $\alpha_{\max}$.

Sowohl um obige Fehlerabschätzung zu erhalten, als auch zur eigentlichen Berechnung des Lotes muß zuerst **E** in die Form **DQ** überführt werden. Das hierfür am geeignetesten erscheinende Verfahren ist die Singuläre-Werte-Zerlegung. Da die Zeitkomplexität des verwendeten Algorithmus aus [34] zur Zerlegung einer Matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ bei $O(4mn^2 + 8n^3)$ liegt, lohnt es sich, über eine Reduktion dieser Berechnungen ebenfalls nachzudenken. Eine einfache Caching-Strategie, die nur nach einer Modifikation von **E** die Werte **D** und **Q** neu berechnet, erweist sich als bereits ausreichend, da die Zahl der Modifikationen wesentlich geringer ist als die Zahl der notwendigen Distanzberechnungen.

Anhand mehrerer Beispieldatensätze aus [106] wurde die Effizienz der Fehlerabschätzung und der Effekt der Caching-Strategie getestet. Tabelle 5.1 zeigt das Ergebnis des Testlaufs. Es wurden 4 Datensätze mit unterschiedlich dimensionalen Eingaberäumen getestet. Während eines On-Line-Lerndurchlaufes wurde die Zahl der benötigten Distanzberechnungen erfaßt. Zusätzlich wurde protokolliert, wie häufig aufgrund der angeforderten Distanzberechnung ein Optimiererlauf und eine Singuläre-Werte-Zerlegung notwendig wurde. Wie sich ablesen läßt, konnte in über 20% der Fälle mit Hilfe der Fehlerabschätzung ein aufwendiger Optimiererlauf vermieden werden. Die Caching-Strategie hat darüberhinaus dazu geführt, daß der Anteil der Singuläre-Werte-Zerlegungen weit unter 1‰ gesunken ist.

Dimension	2	4	6	12
Distanzberechnungen	13556602	49175766	54966002	53466779
Optimierungsläufe	17.09%	13.56%	11.01%	15.74%
Singuläre-Werte-Zerlegungen	0.1682‰	0.02556‰	0.02773‰	0.04081‰

Tabelle 5.1: Effizienz der Fehlerabschätzung an einem Beispieldatensatz aus [106]

Die hier eingeführte Fehlerabschätzung führt also in Kombination mit dem Caching der Singuläre-Werte-Zerlegung zu einer drastischen Geschwindigkeitssteigerung. Somit kann das korrekte Abstandsmaß trotz der erhöhten Berechnungskomplexität eingesetzt werden und sorgt für eine wesentliche Verbesserung in der Klassifikationsgüte des zugrundeliegenden Clusterverfahrens, da die auf Seite 56 definierten Randbedingungen (1)-(3) erfüllt werden. (Siehe auch Abbildung 5.7 auf Seite 62)

5.2 On-Line-Clustering mit Ellipsoiden Karten

In diesem Abschnitt wird das zugrunde liegende Clusterverfahren erläutert, das die ellipsoide Kartierung des Prozeßzustandsraumes vornimmt. Es handelt sich im Wesentlichen um eine konsequente Zusammenführung und Weiterentwicklung der bereits in [99] vorgestellten Verfahren zur *„Kartierung bekannter Teile des Konfigurationsraums von Manipulatoren an Hand von Beispieltrajektorien mittels Ellipsoidkarten“* [49, 11, 7] und der *„Selbstorgansierenden Prozeß-Zustandserkennung“* [95].

Der Grundgedanke des ellipsoiden On-Line-Clustern ist es, unbekannte Prozeßzustände beim Auftreten in ein vorhandenes Codebuch inkrementell aufnehmen zu können. Hierbei wird ein um ellipsoide Karten erweitertes Codebuch genutzt, so daß eine Unterscheidung zwischen bereits bekannten und noch unbekannten Regionen des Prozeßzustandsraumes vorgenommen werden kann. Ein Codebucheintrag besteht dann nicht mehr nur aus dem Codebuchvektor $\vec{c}_i$ sondern aus dem Tupel $\{\vec{c}_i, \mathbf{E}_i, \mathfrak{D}_i, \mathfrak{M}_i, K_i\}$ wobei durch $\{\vec{c}_i, \mathbf{E}_i, \mathfrak{D}_i\}$ wie in Abschnitt 5.1.2 eingeführt ein Hyperellipsoid definiert wird. Zusätzlich ist diesem noch ein lokales Modell $\mathfrak{M}_i$ für Vorhersagen $\mathfrak{M}_i(\vec{x})$ im zugehörigen Prozeßzustandsraum, und die Konfidenz K_i zugeordnet, welche eine Aussage über die Vorhersagegüte von $\mathfrak{M}_i$ zuläßt.

Für einen gemessenen Eingabe-Datenvektor $\vec{x}$ wird dasjenige k gesucht, für das laut Gleichung (5.5) die Zugehörigkeit $\vec{x} \in \{\vec{c}_k, \mathbf{E}_k\}$ erfüllt ist. Anschließend wird er in die zwei Kategorien *bekannt* und *unbekannt* eingeteilt. Hierbei gilt:

$$\vec{x} \text{ ist } \textit{bekannt} \;\Leftrightarrow\; \exists k : \vec{x} \text{ liegt innerhalb von } \{\vec{c}_k, \mathbf{E}_k\} \;\Leftrightarrow\; \exists k : \|\mathbf{E}_k(\vec{x} - \vec{c}_k)\| \leq 1$$

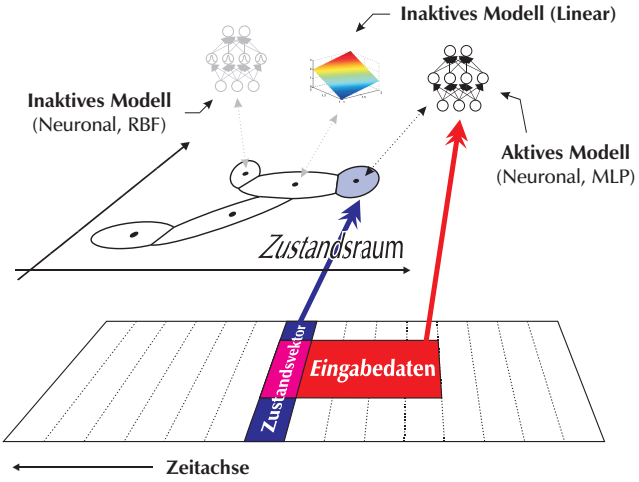


Abbildung 5.11: Verwendung ellipsoider Karten zur Zustandserkennung gekoppelt mit anschließender Modellvorhersage

Das eigentliche Clustern startet mit einem leeren Codebuch. Treten *bekannte* Prozeßzustände auf, so wird vom zuständigen lokalen Modell $\mathfrak{M}_k$ eine Modellvorhersage getroffen, wie in Abbildung 5.11 gezeigt.

Weicht der real gemessene Ausgabe-Datenvektor $\vec{y}$ um weniger als ϵ von der Vorhersage ab, gilt also $\|\mathfrak{M}_k(\vec{x}) - \vec{y}\| < \epsilon$, so wird dem lokalen Modell $\mathfrak{M}_k$ das Ein-/Ausgabepaar $(\vec{x}, \vec{y})$ zum Training angeboten. Bei einer zu großen Abweichung wird anhand der aktuellen Konfidenz K_k entschieden, wie weiter vorgegangen werden soll:

$K_{\text{bad}} \leq K_k$ — Der Vorhersage-Fehler wird als Zeichen gewertet, daß die lokale Dynamik noch nicht gänzlich gelernt ist. Da bisher gute Vorhersagen gemacht wurden, ist es sinnvoll, das Ein-/Ausgabepaar $(\vec{x}, \vec{y})$ dem lokalen Modell $\mathfrak{M}_k$ zum Nachtrainieren anzubieten.

$K_{\text{del}} \leq K_k < K_{\text{bad}}$ — Der Vorhersage-Fehler wird als systematisch angesehen und das Ein-/Ausgabepaar $(\vec{x}, \vec{y})$ wird zur späteren Bearbeitung in die sogenannte Ausnahmen-Liste aufgenommen. Hierdurch können im folgenden Subellipsoide generiert werden, die Sonderfälle innerhalb anderer Ellipsoide abdecken.

$K_k < K_{\text{del}}$ — Es wird wie im vorherigen Fall verfahren, zusätzlich wird der Codebucheintrag k gelöscht. Nachdem in der Regel die Konfidenz eines Modells langsam fällt, werden durch die vorherige Regel zuerst Subellipsoide angelegt um Sonderfälle abzufangen. Somit übernehmen diese die Funktion des gelöschten Eintrags. Auf diese Weise wird ein Ellipsoid automatisch gesplittet, falls es erforderlich ist.

Die *unbekannten* Prozeßzustände werden normalerweise in der Ausnahmen-Liste gesammelt. Ist aber das nächste Ellipsoid weniger als $d_{\min}$ von $\vec{x}$ entfernt und macht das zugehörige

Modell eine ausreichend gute Vorhersage, gilt also $\mathfrak{D}_k(\vec{x}) \leq d_{\min} \wedge \|\mathfrak{M}_k(\vec{x}) - \vec{y}\| < \epsilon$, so wird das Ellipsoid des Codebucheintrags in Richtung von $\vec{x}$ erweitert und $\vec{x}$ als *bekannt* behandelt.

In der sogenannten Ausnahmen-Liste werden Prozeßzustände gesammelt, bis ein festgelegtes Kriterium erfüllt ist, normalerweise das Erreichen der maximal erlaubten Zahl von Ausnahmen. Dann wird die Ausnahmen-Liste mit Hilfe des OLVQ1-Algorithmus aus [53, 51, 52] geclustert, der folgendermaßen definiert ist:

repeat N **times**
  for Alle Datenvektoren $\vec{x}$
    Finde k mit $\forall i \neq k : \|\vec{m}_i - \vec{x}\| \geq \|\vec{m}_k - \vec{x}\|$
    if $\vec{x}$ gehört zur Klasse k
      Ersetze $\vec{m}_k$ durch $\vec{m}_k + \alpha_k(\vec{x} - \vec{m}_k)$
      Ersetze α_k durch $\frac{\alpha_k}{1+\alpha_k}$
    else
      Ersetze $\vec{m}_k$ durch $\vec{m}_k - \alpha_k(\vec{x} - \vec{m}_k)$
      Ersetze α_k durch $\frac{\alpha_k}{1-\alpha_k}$
    end if
  end for
end repeat

Da in diesem Spezialfall nur Datenvektoren eines Prozeßzustandsraumes geclustert werden, steht in der Regel keine Klasseninformation zur Verfügung. Der ursprüngliche Algorithmus vereinfacht sich also zu:

repeat N **times**
  for Alle Datenvektoren $\vec{x}$
    Finde k mit $\forall i \neq k : \|\vec{m}_i - \vec{x}\| \geq \|\vec{m}_k - \vec{x}\|$
    Ersetze $\vec{m}_k$ durch $\vec{m}_k + \alpha_k(\vec{x} - \vec{m}_k)$
    Ersetze α_k durch $\frac{\alpha_k}{1+\alpha_k}$
  end for
end repeat

Die so gewonnen neuen Prozeßzustände bzw. Clusterzentren $\vec{m}$ werden anschließend mit Hilfe eines zweiten Clusterverfahrens in das Codebuch eingefügt. Hierbei wird die Lage der neu zu generierenden zu den bereits bestehenden Ellipsoiden und deren Konfidenz berücksichtigt:

Befindet sich ein neues Clusterzentrum $\vec{m}$ innerhalb bereits kartierten Raumes, ist also $\vec{m}$ *bekannt*, so wird der zuständige Codebucheintrag $\{\vec{c}_k, \mathbf{E}_k, \mathfrak{D}_k, \mathfrak{M}_k, K_k\}$ auf Unterschreiten der Konfidenzschwelle K_{bad} getestet. Wenn K_k zu niedrig liegt, muß die durch das momentan zugeordnete Modell $\mathfrak{M}_k$ unzureichende Modellierung verbessert werden. Dies erfolgt entweder durch Einfügen eines Subellipsoids innerhalb von $\{\vec{c}_k, \mathbf{E}_k\}$ oder durch räumliche Adaption eines bestehendes Subellipsoids. Sollte die Konfidenz K_k ausreichend sein, so wird $\vec{m}$ ignoriert.

Liegt ein neues Clusterzentrum $\vec{m}$ hingegen außerhalb des kartierten Raumes, so wird ein kreisförmiges Ellipsoid mit initialem Radius r_0 generiert bzw. ein bestehendes Ellipsoid räumlich adaptiert.

Abschließend wird ein eventuell erzeugtes Ellipsoid bzw. Subellipsoid noch mit einem neu-

en Modell $\mathfrak{M}$ verknüpft, das anhand des in Abschnitt 5.3.2 erläuterten Algorithmus selektiert und initialisiert wird.

5.2.1 Räumliche Adaption von Ellipsoiden

Jeder Codebucheintrag k enthält ein Ellipsoid, welches durch das Zentrum $\vec{c}_k$ und die Matrix $\mathbf{E}_k$ beschrieben wird. Der Rand des Ellipsoids besteht offensichtlich aus allen Punkten $\vec{p}$, für die $\|\mathbf{E}_k(\vec{p}-\vec{c}_k)\| = 1$ gilt.

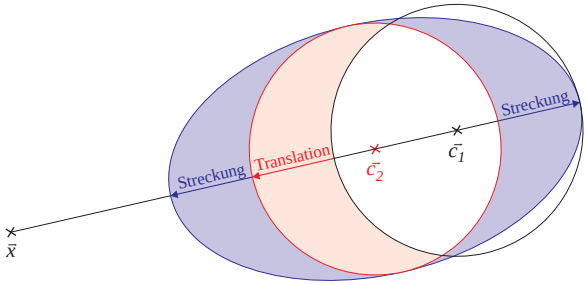


Abbildung 5.12: Das Anpassen eines Ellipsoids an einen neuen Datenpunkt $\vec{x}$ erfolgt durch Verschiebung von $\vec{c}$ und Streckung von $\mathbf{E}$.

Sei jetzt $\vec{x}$ ein Punkt in dessen Richtung das Gebiet eines Ellipsoids erweitert werden soll. Die Adaption findet statt, indem das Gebiet ein wenig in Richtung von $\vec{x}$ verschoben wird und danach das Ellipsoid entlang der Verschiebungsrichtung durch lineare Streckung ein wenig vergrößert wird. Abbildung 5.12 zeigt an einem 2-dimensionalen Beispiel, daß so der alte Einflußbereich im wesentlichen erhalten bleibt. Die neuen Werte[2] $\vec{c}_k{}'$ und $\mathbf{E}_k{}'$ ergeben sich laut [7, 11] folgendermaßen:

$$\begin{aligned}\vec{c}_k{}' &= \vec{c}_k + \alpha(\vec{x}-\vec{c}_k)\\ \mathbf{E}_k{}' &= \mathbf{E}_k + (\beta-1)\vec{a}\vec{a}^T\mathbf{E}_k\end{aligned} \quad \text{mit} \quad \beta = \frac{1}{1+\|\alpha\mathbf{E}_k(\vec{x}-\vec{c}_k)\|},\ \vec{a} = \frac{\mathbf{E}_k(\vec{x}-\vec{c}_k)}{\|\mathbf{E}_k(\vec{x}-\vec{c}_k)\|} \tag{5.19}$$

Der Parameter $\alpha \in [0,1]$ gibt an, wie weit verschoben werden soll. Für seine Wahl gibt es verschiedene Verfahren: Stets die gleiche (kleine) Konstante oder so, daß $\vec{x}$ nach der Adaption genau auf dem Rand liegt oder so, daß $\vec{c}_k$ genau dem Schwerpunkt der bisher adaptierten Punkte entspricht. Die Streckung entlang des normierten Verschiebungsvektors $\vec{a}$ mit geeigneter Wahl von β garantiert, daß der $\vec{x}$ gegenüberliegende Randpunkt auf der Verschiebungsachse fest bleibt.

Basierend auf obiger Adaptionsformel läßt sich bereits ein einfacher Cluster-Algorithmus definieren. Dieser verwendet die in Abschnitt 5.1.2 eingeführte ellipsoide Variante $\mathfrak{D}$ als Abstandsmaß. Aus der Zugehörigkeit von $\vec{x}$ und dem Vorhersage-Fehler $\|\mathfrak{M}_k(\vec{x})-\vec{y}\|$ ergibt sich

[2] Im weiteren Verlauf wird die Schreibweise x bzw. x' synonym zu $\mathrm{x}(t_k)$ bzw. $\mathrm{x}(t_{k+1})$ verwendet, wobei t_{k+1} den nächsten Zeitschritt nach t_k bezeichnet.

dabei die Klasseninformation:

repeat N **times**
  for Alle Datenvektoren $\vec{x}$
    Finde k mit $\forall i \neq k : \mathfrak{D}_i(\vec{x}) \geq \mathfrak{D}_k(\vec{x})$
    if $\vec{x}$ ist *unbekannt* $\wedge \; \|\mathfrak{M}_k(\vec{x}) - \vec{y}\| \leq \epsilon$
      Ersetze $\vec{c}_k$ durch $\vec{c}_k + \alpha_k(\vec{x} - \vec{c}_k)$
      Ersetze $\mathbf{E}_k$ durch $\mathbf{E}_k + (\beta - 1)\vec{a}\vec{a}^T\mathbf{E}_k$
      Optional: Modifiziere α_k nach gewünschter Methode
    end if
  end for
end repeat

Man erkennt starke Ähnlichkeiten zu dem zuvor beschriebenen OLVQ1-Algorithmus. Einerseits wurde bei der Suche des nächsten Nachbarn nur das euklidische Abstandsmaß durch die ellipsoide Variante $\mathfrak{D}$ ersetzt. Andererseits wird das Fehlen der Klasseninformation durch einen Test auf Zuständigkeit und Güte der Vorhersage kompensiert.

Die im OLVQ1-Algorithmus erfolgte Adaption der „Lernrate“ α entspricht exakt der bereits erwähnten Vorgehensweise, das Zentrum $\vec{c}$ dem Schwerpunkt der bisher adaptierten Punkte folgen zu lassen. Sei m_k die Summe der „Massen“ aller Punkte an die bisher adaptiert wurde. Legt man die „Masse“ eines Punktes mit Eins fest, so entspricht m_k der Zahl der bisher erfolgten Adaptionen von $\vec{c}_k$. Es gilt dann $m'_k = m_k + 1$ und man erhält:

$$\begin{aligned} \vec{c}_k + \alpha_k(\vec{x} - \vec{c}_k) = \frac{m_k\vec{c}_k + \vec{x}}{m_k + 1} \quad &\Rightarrow \quad \alpha_k = \frac{1}{m_k + 1},\; m_k = \frac{1 - \alpha_k}{\alpha_k} \\ &\Rightarrow \quad \alpha'_k = \frac{1}{m'_k + 1} = \frac{1}{m_k + 2} = \frac{\alpha_k}{\alpha_k + 1} \end{aligned} \tag{5.20}$$

Der komplette Algorithmus für das On-Line-Clustering ergibt sich durch Erweiterung dieses Prototyps um Konfidenz- und Ausnahmenbehandlung. Der erweiterte Algorithmus mit dem Namen LEMON[3] findet sich im Anhang ab Seite 115. Das Ergebnis eines On-Line-Cluster-Laufes von LEMON mit Hysterese-Daten ist in Abbildung 5.13 gezeigt.

Im bisherigen Verlauf wurde bereits mehrfach Bezug auf das einem Cluster zugeordnete lokale Modell $\mathfrak{M}$ und die Konfidenz K genommen. Der folgende Abschnitt soll nun klären, wie im einzelnen ein lokales Modell ausgewählt und initialisiert wird. Auch die Berechnung der Konfidenz und deren Bedeutung wird näher erläutert. Ein Vergleich globaler und lokaler Modellbildungsmethoden soll zu Beginn den Nutzen von LEMON weiter verdeutlichen.

5.3 Die lokale Modellbildungskomponente

Lokale Modellierungmethoden verwenden in der Regel wenig komplexe, nur lokal gültige Modelle. Der Begriff lokal gültige Modelle zielt dabei auf den Ursprungsraum des Modellierungsproblems ab. Man unterscheidet bei lokalen Modellierungmethoden wiederum zwischen lokalem und globalem Lernen. Lokales Lernen garantiert lokal beschränkte Parametermodifikationen, so daß nur die lokalen Modellierungseigenschaften verändert werden. Globales Lernen

[3] **L**ocal **E**llipsoidal **Mo**del **N**etwork

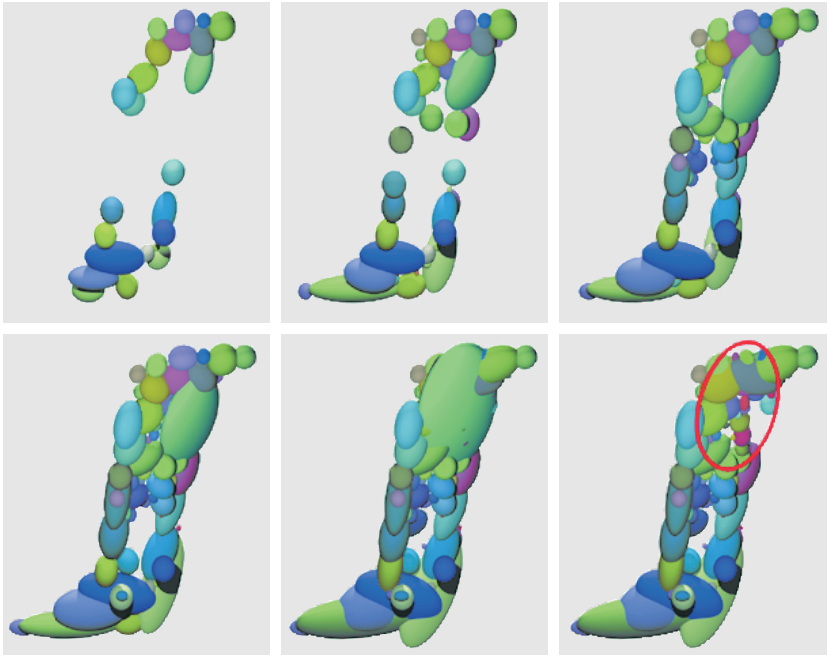


Abbildung 5.13: Hier ist als Beispiel eines On-Line-Cluster-Laufes das Lernen einer Hysterese dargestellt. Man erkennt zum einen das inkrementelle Hinzufügen neuer Clusterzentren, zum anderen das Adaptieren bereits vorhandener Ellipsoide. Das Löschen eines „schlechten“ Clusters ist in den letzten beiden Abbildungen sichtbar. Dabei ist der bisher vom gelöschten Ellipsoid eingenommene Raum mit einem roten Rand markiert. Wie vorgesehen, befinden sich bereits Subellipsoide innerhalb dieses Raumes und übernehmen die Modellierungsaufgabe des gelöschten Clusters.

bei lokaler Modellbildung ist möglich, wenn zwischen den lokalen Modelle nicht „hart“ umgeschaltet, sondern über eine Maskierungsfunktion – wie etwa der Gaußschen Glockenfunktion bei RBF-Netzen [75] – „weich“ übergeblendet wird. Betrachtet man dann das Gesamtsystem, so lassen sich globale Lernverfahren anwenden, die gleichzeitig alle Parameter des Systems modifizieren.

Globale Modellierungsmethoden zeichnen sich dagegen durch komplexe, global gültige Modelle aus. Ein klassisches Beispiel aus der Theorie der neuronalen Netze ist das Multi-Layer-Perzeptron [84], kurz MLP genannt. Diese Weiterentwicklung des von Rosenblatt [82] vorgestellten Perzeptrons verfügt über eine stark verschränkte Informationsverarbeitung, so daß keine lokale Zuordnung von Modell-Parametern[4] zu Eingaberegionen möglich ist. Wird in diesem System ein Parameter variiert, so wirkt sich die Variation auf alle bisher gelernten Informationen aus. In der Regel wird deshalb ein MLP im Batch-Verfahren mit zufällig gemischten Lerndaten trainiert.

Es ist offensichtlich, daß sich On-Line-Lernen und globale Modellbildung widersprechen. In LEMON wurde also konsequenterweise ein lokaler Ansatz verwendet.

Von verschiedenen Autoren [67, 101] werden lokale Modelle und Lernverfahren propagiert, um Probleme mit globalen Methoden zu umgehen. Im folgenden sollen einige Vor- und Nachteile lokaler und globaler Methoden angesprochen werden:

Modifikation globaler Information. In den meisten Anwendungsfällen ist es problematisch, wenn durch Lernen neuer Datenvektoren bereits erworbenes Wissen aufgrund von Interferenzen wieder verloren geht. Lokale Methoden bieten hier im Gegensatz zu globalen Verfahren die Sicherheit, daß Modifikationen nur lokale Auswirkungen haben. Eine ähnliche Problematik im Zusammenhang mit Parameteradaption ist in [4] beschrieben.

Generalisierungsfähigkeit. Hier wirkt sich die Verwendung lokaler Verfahren nachteilig aus, da Generalisierung nur im Rahmen der lokalen Zuständigkeit möglich ist. Da jedes lokale Modell nur für einen relativ kleinen Bereich zuständig ist und somit relativ wenig freie Parameter hat, kann es während des On-Line-Lernens früh beginnen, lokal zu generalisieren.

Rechenzeit. In der Regel können aufgrund der reduzierten Zuständigkeit sehr kleine lokale Modelle verwendet werden, was einen Rechenzeitvorteil bringt, der oftmals – je nach Aufteilung des Eingaberaumes – durch die Suche nach dem zuständigen Modell wieder zunichte gemacht wird.

Allgemein läßt sich bemerken, daß durch die Verwendung lokaler Methoden sowohl die Struktur als auch die Komplexität eines Modells an unterschiedliche Gegebenheiten angepaßt werden kann, abhängig von der Position im Eingaberaum. In wenig dynamischen Umgebungen können also lineare oder andere leicht handhabbare, in stark dynamischen aber auch hochkomplexe Modelle eingesetzt werden. Anhand der automatisch gewählten lokalen Modellstruktur sind Rückschlüsse auf die Komplexität der realen Dynamik in diesem Eingabebereich möglich. Im Gegensatz dazu muß ein globales Verfahren immer so komplex gewählt werden, wie die komplexeste Modellierungsaufgabe es diktiert.

[4] Beim MLP werden die Modell-Parameter üblicherweise als Gewichte bezeichnet, welche die einzelnen Modellschichten miteinander verknüpfen. Siehe auch Kapitel 3.3, Seite 24.

In LEMON werden alle derzeit in der Modellbibliothek AMoC [100] implementierten Modelle unterstützt. Hierzu zählen unter anderem lineare Modelle, MLP und RBF-Netze.

5.3.1 Das Konfidenzmaß

Für verschiedene Anwendungen ist es wichtig zu wissen, inwieweit der Vorhersage eines neuronalen Netzes vertraut werden kann, es wird also ein Konfidenzmaß benötigt. Dies ist insbesondere wichtig, wenn das neuronale Netz vor Ort in der Anwendung On-Line (nach-) trainiert wird. Beispielsweise kann man neuronale Netze zur Modellbildung innerhalb eines intelligenten Regelungssystems unter anderem folgendermaßen einsetzen: Zunächst wird ein konventioneller Regler benutzt, der die Regelungsaufgabe zwar stabil und zuverlässig löst, aber keine optimale Performanz bieten kann, da die Regelstrecke a priori nicht ausreichend bekannt ist. Normalerweise wird nun ein neuronales Netz parallel zum laufenden Prozeß mit der inversen Dynamik belernt und als Vorsteuerung eingesetzt [43].

Dieser Ansatz hat in der Regel diverse Nachteile. Der wohl gravierendste ist mangelnde Stabilität. Dies resultiert aus der Tatsache, daß speziell neuronale Netze in noch unbekannten Regionen des Eingaberaumes völlig zufällige Ergebnisse bei der Vorhersage liefern. Zur stabilen Regelung eines Prozesses mit Hilfe der inversen Modellierung benötigt man also zusätzlich eine Aussage über die Zuverlässigkeit der gemachten Vorhersage. Bei schlechten Vorhersagen kann so auf den konventionellen Regler umgeschaltet werden, der dann zwar keine optimale Performanz erreicht, aber Stabilität garantiert.

Es sind mehrere Möglichkeiten zur Schätzung der Zuverlässigkeit einer Vorhersage denkbar:

Bayes-Modelle. Aufgrund der expliziten Darstellung als Wahrscheinlichkeitsmodell liefert ein neuronales Netz auf Basis eines Bayes-Modells keinen einzelnen Wert als Ergebnis, sondern eine Verteilung, die somit implizit auch die Konfidenz eines bestimmten Ergebnisses enthält. Leider sind diese Modelle immer noch sehr rechenaufwendig, da Integrale über mehrere Dimensionen berechnet werden müssen. Selbst durch Monte-Carlo-Methoden vereinfachte Modelle sind noch zu komplex, um in LEMON genutzt zu werden [73].

Neuronale Netze als Konfidenz-Schätzer. Häufig trifft man auch auf neuronale Netze zur Konfidenzschätzung wie zum Beispiel in [87]. Der Vorteil dieses Ansatzes ist seine Adaptivität. Problematisch ist, daß man Neuronale Netze verwendet, um Neuronale Netze zu beurteilen. Denkt man dies zu Ende benötigt man eine unendlich Zahl von „Überwachern“, um „sicher“ zu sein. Gerade da man viele Arten von neuronalen Netze im On-Line-Betrieb nicht ausreichend analysieren kann, will man die Konfidenz mit *anderen* Verfahren, die analysierbar sind, überwachen.

Heuristiken. Intuitiv am einsichtigsten sind Heuristiken, die anhand einfacher Statistiken auch modellübergreifend realisiert werden können. So kann aus der Zahl der bisherigen korrekten und falschen Vorhersagen ein Maß für die Wahrscheinlichkeit einer korrekten Vorhersage abgeleitet werden. Der Nachteil dieser Methode ist, daß bei globalen Modellen pro Eingabevektor eine separate Statistik angelegt werden muß. Eine globale, von der jeweiligen Eingabe unabhängige, Heuristik bringt keinen zusätzlichen Nutzen, da sie nur Aussagen über die allgemeine Vorhersagegüte macht. Meist ist die globale Gültigkeit der Heuristik nicht oder nur empirisch verifizierbar.

Untersucht man die drei genannten Varianten der Konfidenzschätzung unter dem Gesichtspunkt der lokalen Modellierung, wie sie in LEMON realisiert ist, so erscheint die Nutzung einer Heuristik *pro* lokalem Modell als die beste Wahl. Da einfache lokale Modelle mit eingeschränktem Einzugbereich genutzt werden, verschwinden auch die Nachteile der globalen Varianten. Man verfügt somit über eine, auf einen bestimmten Bereich des Zustandsraumes bezogene Konfidenzaussage. In [87] werden zum Training eines Konfidenzmoduls zwei Arten von Heuristiken genannt. Einerseits die Wahrscheinlichkeit einer korrekten Vorhersage, geschätzt durch den Anteil der korrekten Vorhersagen an allen bisher getätigten Vorhersagen. Andererseits die erwartete Abweichung vom korrekten Wert, geschätzt durch die mittlere bisherige Abweichung. Beide Maße lassen die Zahl der bisherigen Trainingsläufe unberücksichtigt.

Da in LEMON das Löschen lokaler Modelle durch deren Konfidenz gesteuert wird, ist es nötig auch die Zahl der bisherigen Trainingsläufe mit zu berücksichtigen. Es ist nicht sinnvoll ein Modell zu behalten, das trotz kontinuierlichen Trainings nicht mehr wesentlich besser wird. In LEMON wird deshalb zur Konfidenzschätzung folgende Heuristik verwendet, die sowohl den mittleren bisherigen Fehler als auch die Zahl der Trainingsschritte berücksichtigt:

$$K = e^{-\gamma N_{\text{Trained}} \overline{E}_{\text{Predicted}}} \tag{5.21}$$

Dabei handelt es sich bei N_{Trained} um die Gesamtzahl der bisher gesehenen Trainingspaare. Wurde noch nicht trainiert, so ergibt sich eine Konfidenz von Eins. Der mittlere Vorhersage-Fehler $\overline{E}_{\text{Predicted}}$ des Modells berücksichtigt Vorhersagen, für die $\|\mathfrak{M}(\vec{x}) - \vec{y}\| < \epsilon$ gilt, als Vorhersagen mit Fehler Null. Mit Hilfe des Parameters γ kann die Sensitivität bezüglich der Zahl der Trainingsbeispiele bzw. der Skalierung des Ausgaberaumes eingestellt werden. Allerdings ist darauf zu achten, daß derselbe Effekt durch Verschieben der Konfidenzgrenzen K_{bad} und K_{del} bzw. Umskalieren des Ausgaberaumes[5] erreicht werden kann, weshalb γ momentan fest auf den Wert Eins gesetzt ist.

Eine nützliche Eigenschaften der in Gleichung (5.21) gewählten Heuristik für K erlaubt eine Interpretation als Wahrscheinlichkeit und lautet:

$$N_{\text{Trained}}, \overline{E}_{\text{Predicted}} \in [0, \infty[\implies \int_{N_{\text{Trained}}, \overline{E}_{\text{Predicted}}} e^{-N_{\text{Trained}} \overline{E}_{\text{Predicted}}} \equiv 1 \tag{5.22}$$

Speziell kann ein übergeordnetes System anhand von Formel (5.21) entscheiden, ob die Vorhersage genutzt oder verworfen werden soll. Somit kann zwischen LEMON und z.B. einem klassischem System umgeschaltet werden, was einen stabilen „Notfallbetrieb“ gewährleistet.

Nachdem eine Möglichkeit geschaffen wurde, die Konfidenz eines lokalen Modells zu beurteilen, wird im folgenden die Auswahl der einzelnen Modelle anhand verschiedener Kriterien erläutert.

5.3.2 Die Modellauswahlkomponente

Wird ein neues Ellipsoid angelegt, so bedeutet dies, daß entweder bisher unbekannte Prozeßzustände oder Sonderfälle innerhalb existierender Ellipsoide erkannt wurden. In beiden Fällen

[5] Aus numerischen Gründen ist es günstig, den Ein- und Ausgaberaum auf das Intervall $[-1, 1]$ zu normieren.

ist es notwendig ein neues lokales Modell für diese bestimmte ellipsoide Region auszuwählen. Für diesen Zweck wurde in LEMON ein Modul integriert, das aus einer Menge vorgegebener Modell-Prototypen das „Beste“ auswählt.

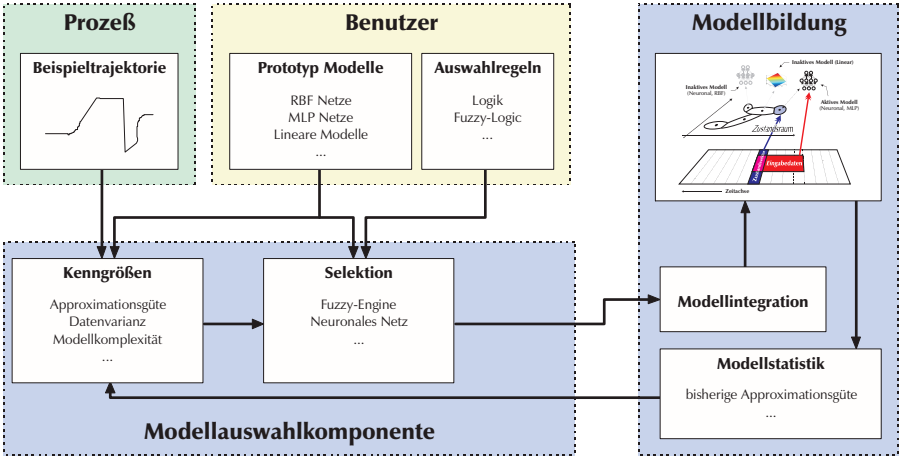
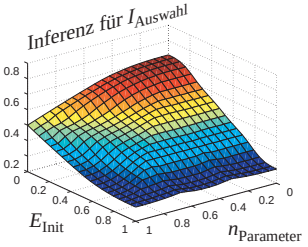


Abbildung 5.14: Kombination von ellipsoiden Karten und lokaler Modellbildung mit Benutzergesteuerter Modellauswahl.

Der Ablauf eines solchen Auswahlvorgangs ist in Abbildung 5.14 schematisch gezeigt: Durch einen Benutzer wird eine beliebige Menge verschiedener Modell-Prototypen mit modellspezifischen Optimierungsverfahren, sowie eine Regelbasis zur Auswahl dieser Modelle vorgegeben. Basierend auf verschiedenen Kenngrößen, die aus der Prozeßtrajektorie, den Modell-Prototypen und dem bisherigen Verlauf des Modellierungsprozesses gewonnen werden, kann dann anhand der Regelbasis für eine gegebene ellipsoide Region ein lokales Modell ausgewählt werden.

Um eine einfache und intuitive Spezifikation der Regeln durch den Benutzer zu ermöglichen, wurde ein Fuzzy-Logik Ansatz [55] zur Spezifikation der Regelbasis gewählt. Hierbei kann sowohl auf Kenngrößen der einzelnen Modell-Prototypen, als auch auf Eigenschaften der in der Ausnahmen-Liste gespeicherten Ein-/Ausgabepaare in Form von linguistischen Variablen zurückgegriffen werden. So steht zum Beispiel für jeden Modell-Prototypen die Zahl der Modell-Parameter n_{Param} und der nach der Modellinitialisierung mittels Crossvalidation (siehe Seite 19) berechnete lokale Approximationsfehler E_{Init} zur Verfügung. Vor der Fuzzyfizierung wird der jeweilige Wertebereich auf $[0, 1]$ normiert, um diese Größen sinnvoll als Fuzzy-Wert nutzen zu können. Für den Modell-Prototypen mit den wenigsten Parametern gilt dann $n_{\text{Param}} = 0$, für den mit den meisten $n_{\text{Param}} = 1$. Das in LEMON eingesetzte Fuzzy-Modul wurde so entworfen, daß es zu FOX [71], einem frei erhältlichen Fuzzy-Inferenzprogramm, kompatibel ist. Das zu FOX gehörige Designtool FOOL kann genutzt werden, um beliebige Regelbasen graphisch zu entwerfen. Das Fuzzy-Modul liefert für jeden Modell-Prototypen ein Inferenzergebnis $I_{\text{Auswahl}} \in [0, 1]$ zurück. Der Prototyp mit dem höchsten Wert von I_{Auswahl} wird schließlich selektiert. Wird keine Regelbasis durch den Benutzer vorgegeben, so findet

die Auswahl der Modelle nur anhand des Fehlers E_{Init} statt. Abbildung 5.15 zeigt das Beispiel einer Fuzzy-Regelbasis, die einfache Modelle bevorzugt.



if $n_{\text{Param}} = \text{groß} \wedge E_{\text{Init}} = \text{groß}$ **then** $I_{\text{Auswahl}} = \text{nein}$
if $n_{\text{Param}} = \text{klein} \wedge E_{\text{Init}} = \text{klein}$ **then** $I_{\text{Auswahl}} = \text{ja}$
if $n_{\text{Param}} = \text{mittel} \wedge E_{\text{Init}} = \text{mittel}$ **then** $I_{\text{Auswahl}} = \text{vielleicht}$
if $n_{\text{Param}} = \text{klein} \wedge E_{\text{Init}} = \text{mittel}$ **then** $I_{\text{Auswahl}} = \text{vielleicht}$
if $n_{\text{Param}} = \text{klein} \wedge E_{\text{Init}} = \text{groß}$ **then** $I_{\text{Auswahl}} = \text{nein}$
if $n_{\text{Param}} = \text{mittel} \wedge E_{\text{Init}} = \text{groß}$ **then** $I_{\text{Auswahl}} = \text{nein}$
if $n_{\text{Param}} = \text{mittel} \wedge E_{\text{Init}} = \text{groß}$ **then** $I_{\text{Auswahl}} = \text{nein}$

Abbildung 5.15: Beispiel einer Fuzzy-Regelbasis, die einfache Modelle bevorzugt. Links ist das defuzzyfizierte Inferenzergebnis I_{Auswahl}, rechts die zugrunde liegende Regelbasis zu sehen. Der lokale Approximationsfehler E_{Init} wird nichtlinear mit der Zahl der Parameter n_{Param} gewichtet: Ist E_{Init} groß, so hat die Parameterzahl n_{Param} keinen Einfluß auf das Inferenzergebnis I_{Auswahl}. Je kleiner E_{Init} wird, desto stärker wird n_{Param} berücksichtigt.

Wie in Abschnitt 3.2.1 erläutert, gibt es viele Kriterien, die eine „gutes“ Modell auszeichnen. Diese sind stark Problemabhängig, so daß eine einmalige Festlegung wie sie häufig vorgenommen wird, nicht sinnvoll erscheint. Gerade spezielles Vorwissen eines Experten sollte immer eingebracht werden. Der eben vorgestellte Ansatz ist sehr flexibel erweiterbar. So können beliebige Kenngrößen aus den Daten und Modell-Prototypen generiert werden. Ebenso ist es möglich, das Fuzzy-Modul durch andere Selektionsverfahren zu ersetzen. Denkbar wäre hier ein Neuro-Fuzzy-Ansatz [70], der die Regelbasis anhand des Erfolges des Modellbildungsvorgangs adaptiv an den Prozeß anpaßt.

In diesem Kapitel wurde das On-Line-Modellbildungsverfahren LEMON hergeleitetet. Es eignet sich zur Modellierung von Dynamikwechseln und verhindert das unvorhersehbare Vergessen bereits gelernter Information durch Einsatz lokaler Modelle. Hervorzuheben ist die automatische Expansion des Einzugsbereichs „guter“ Modelle. Im nächsten Kapitel finden sich Simulationsbeispiele, die die Mächtigkeit von LEMON demonstrieren.

Kapitel 6

Simulationsergebnisse

Nachdem die entwickelten Filter- und Modellbildungsverfahrens umfassend beschrieben wurden, soll die Eignung dieser anhand verschiedener Daten getestet werden. Der Schwerpunkt wird dabei auf einer verständlichen und klaren Analyse der erzielten Ergebnisse liegen, um die Möglichkeiten der entworfenen Methoden genau einzugrenzen.

6.1 Regressionsfilter mit Sprungerkennung

Um die Eignung des in Abschnitt 4.1.6 entwickelten Regressionsfilters mit Sprungerkennung zur Filterung stark verrauschter Daten mit einzelnen Ausreißern zu testen wurde der im folgenden beschriebene Versuch durchgeführt. Die anschließenden Versuchsergebnisse werden qualitativ bewertet, da sich bereits kleine Variationen der Filterparameter stark auf eine quantitative Aussage wie etwa den Abstand zum Originalsignal auswirken. Nachdem aber eine Anpassung der Filterparameter bei jeder Messung notwendig ist, kann ein qualitativer Vergleich mehr Information vermitteln als quantitative Aussagen.

6.1.1 Versuchsaufbau

Verschiedene Filter werden mit einem gestörten Signal versorgt. Anschließend wird das gefilterte Signal mit dem Originalsignal verglichen und die Filtereigenschaft bewertet. Das unverrauschte Signal ist durch folgende Funktion gegeben:

$$\mathrm{f}(t) = \begin{cases} \sin(0.25 \cdot t + 10) - \sin(10) & \text{für } t < 5 \\ \sin(0.25 \cdot t + 10) - \sin(10) + 0.3 & \text{für } t \geq 5 \end{cases}$$

Wie zu erkennen ist, verfügt es über einen Signalsprung zum Zeitpunkt $t = 5$. Dem Signal $\mathrm{f}(t)$ wird zusätzlich eine Kombination aus weißem Rauschen, einer sinusförmigen Störung mit Amplitude 0.1 und Frequenz 40Hz und gelegentlichen Ausreißern hinzugefügt. In Abbildung 6.1 ist unter anderem das Originalsignal und in Abbildung 6.2 der resultierende verrauschte Signalverlauf zu sehen.

Die im folgenden getesteten Regressionsfilter wurden zur Nutzung mit linearen Modellen parametriert. Der zur Regression betrachtete Zeitraum lag bei 2 Sekunden, die Abtastrate bei 0.005 Sekunden. Der Konfidenzparameter der Sprungerkennung wurde auf 0.999 eingestellt, die Varianzberechnung erfolgte über einen Zeitraum von 1 Sekunde. Zur Detektion eines Sprungs mußte die Konfidenzgrenze länger als 0.05 Sekunden überschritten werden.

6.1.2 Test der Sprungerkennung

Nun wird die Reaktion eines Regressionsfilters ohne Sprungerkennung auf den Signalsprung getestet, um anschließend die Verbesserung durch Hinzufügen der Sprungerkennung zu demonstrieren.

Resultate

Abbildung 6.1 zeigt das Verhalten eines Regressionsfilters ohne Sprungerkennung. Zum Vergleich ist auch der Verlauf des im nächsten Abschnitt getesteten Chebyshev-Filters mit eingezeichnet. Der gefilterte Signalverlauf liegt nach der notwendigen Einschwingphase nahe am Originalsignal. Während der Einschwingphase ist der Verlauf erst identisch zum verrauschten Signal, um sich mit zunehmender Zahl der zur Verfügung stehenden Meßpunkte dem Originalverlauf anzugleichen.

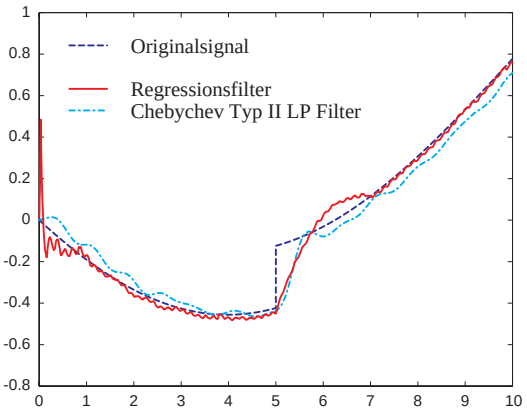


Abbildung 6.1: Vergleich eines Regressionsfilters mit einem analogen Chebyshev Typ II Tiefpaßfilter. Die Grenzfrequenz des Chebyshev-Filters betrug 50Hz, wodurch qualitativ dieselbe Filterung erreicht wird, wie beim Regressionsfilter. Man erkennt das gleiche Anstiegsverhalten nach dem Signalsprung.

Fügt man die Sprungerkennung hinzu, so erhält man den in Abbildung 6.2 gezeigten Verlauf. Durch Überschreiten der eingezeichneten oberen oder unteren Konfidenzgrenze um mehr als 0.05 Sekunden wird die Meßpunkthistorie gelöscht, und der Regressionsfilter folgt unmittelbar dem verrauschten Signalverlauf um sich dann wieder dem Originalsignal anzunähern. Wie zu erkennen ist, bringt die Sprungerkennung in diesem Beispiel eine deutliche Verbesserung des gefilterten Signals. Auch wenn berücksichtigt wird, daß nach dem Dynamikwechsel zuerst ein erhöhtes Rauschen in Kauf genommen werden muß, so erfolgt danach ein wesentlich schnelleres Einschwingen auf die Originaldynamik als ohne Sprungerkennung.

6.1.3 Vergleich mit klassischen Filterverfahren

Mit Hilfe der MATLAB-Toolbox für Signalverarbeitung [102] wurden verschiedene klassische Filter – jeweils die analoge und digitale Variante – und die Regressionsfilter getestet. Die betrachteten Filter waren:

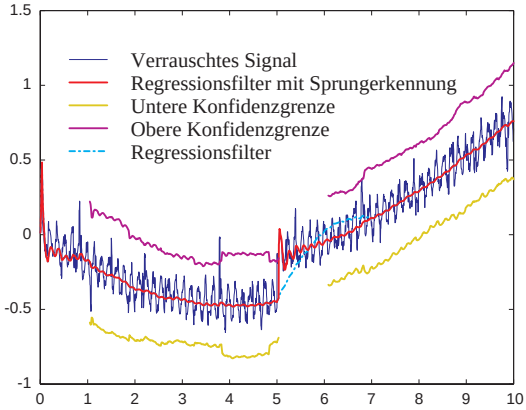


Abbildung 6.2: Vergleich eines Regressionsfilters mit und ohne Sprungerkennung. Der Signalverlauf des gefilterten Signals ist bis auf die Sprungregion bei beiden Filtern identisch. Der Filter ohne Sprungerkennung zeigt das zu erwartende langsame Anstiegsverhalten und geht zum Zeitpunkt $t = 7$ in den Verlauf des Filters mit Sprungerkennung über.

FIR Digitale Filter. Dieser Filtertyp wurde mit verschiedenen Abtastzeiten und Grenzfrequenzen betrieben.

Butterworth, Chebyshev Typ I/II Tiefpassfilter. Sowohl im analogen als auch digitalen Bereich wurde die Ordnungen und Grenzfrequenz variiert. Bei den digitalen Varianten wurde zusätzlich die Abtastzeit geändert.

Regressionsfilter mit/ohne Sprungerkennung. Für beide Filtervarianten ist die Ordnung des Polynomialen Modells, die Zahl der genutzten Regressionspunkte und die Abtastzeit variiert worden. Zusätzlich wurde der Konfidenzparameter der Sprungerkennung angepaßt.

Es hat sich gezeigt, daß unter den klassischen Filtern die analoge Variante des Chebyshev Typ II Filters[1], die besten Ergebnisse erzielen konnte. Stellvertretend für die restlichen Filtertypen soll dieser mit den Regressionsfiltern verglichen werden.

Resultate

Um einen Vergleich zu ermöglichen, wurden für den Chebyshev Typ II Filter zwei unterschiedliche Grenzfrequenzen ermittelt. So tritt einerseits mit 50Hz ein ähnliches Filterverhalten bei Betrachtung des resultierenden Signals bezüglich der Glattheit auf. Andererseits erhält man bei 200Hz einen ähnlichen Anstieg als Reaktion auf den Dynamikwechsel.

Abbildung 6.3 zeigt daß die Performanz des 50Hz Chebyshev-Filters bezüglich des Dynamikwechels schlechter ausfällt als die des Regressionsfilters mit Sprungerkennung. Der Anstieg verläuft deutlich langsamer bei ansonsten etwa gleicher Filterleistung.

[1]Der verwendete Chebyshev Typ II Filter hatte Ordnung 4, und eine Abschwächung der Überschwinger im Sperrbereich von 40 Db.

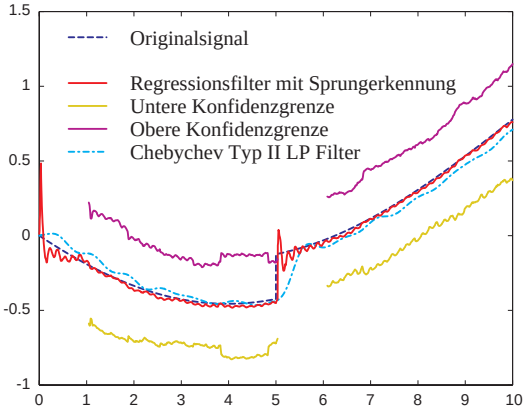


Abbildung 6.3: Vergleich zwischen einem Regressionsfilter mit Sprungerkennung und einem analogen Chebyshev Typ II Tiefpaßfilter mit 50Hz Grenzfrequenz.

Erhöht man die Grenzfrequenz des Chebyshev-Filters um die Reaktion auf den Sprung zu verbessern, so erhält man den in Abbildung 6.4 gezeigten Effekt: Das Rauschen wird weniger stark gefiltert und die Filterperformanz bezüglich Übereinstimmung des gefilterten Signals mit dem Originalsignal sinkt.

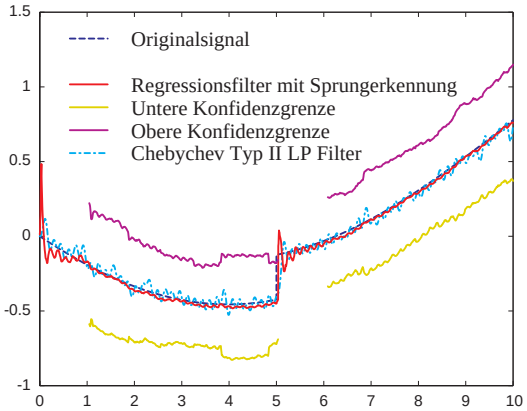


Abbildung 6.4: Vergleich zwischen einem Regressionsfilter mit Sprungerkennung und einem analogen Chebyshev Typ II Tiefpaßfilter mit 200Hz Grenzfrequenz.

6.1.4 Beurteilung

Keiner der betrachteten klassischen Filter konnten einem Dynamikwechsel folgen. Zwar läßt sich die Reaktion in Grenzen anpassen, die Verschlechterung der Filterleistung ist allerdings

beträchtlich. Durch Kombination eines FIR-Filters mit der Sprungerkennung – hier wurde der Regressionsfilter verwendet – ist eine wesentlich bessere Reaktion auf Dynamikwechsel möglich, ohne Einbußen bei der Filterleistung hinnehmen zu müssen. Ein spezieller Vorteil der Regressionsfilter, das gezielte „Vergessen“ der alten Dynamik, führt zu einer schnellstmöglichen Adaption an die neue Dynamik. Der Regressionsfilter mit Sprungerkennung ist somit eine sinnvolle Erweiterung klassischer FIR-Filter.

6.2 Modellbildung mit LEMON

Die in diesem Abschnitt mit dem Modellbildungsverfahren LEMON durchgeführten Simulationen dienen einerseits dazu, die vielfältigen Anwendungsmöglichkeiten zu demonstrieren. Andererseits soll die Tauglichkeit zum On-Line-Lernen überprüft werden. So stammen die zur Verfügung stehenden Daten sowohl aus Simulationen realer Vorgänge als auch von speziell konstruierten Beispielfunktionen. Auch die verschiedenen Möglichkeiten der Analyse, die durch LEMON einen intuitiven Zugang zu den Daten ermöglichen, werden vermittelt.

6.2.1 Künstliche Beispielfunktion

Die hier zugrunde gelegte Beispielfunktion soll zeigen, wie LEMON bei Bedarf zwischen verschiedenen Dynamiken bzw. Modellen wechselt. Zusätzlich wird seine Eignung demonstriert, auch harte Übergänge zu modellieren, wie sie etwa beim Wechseln des Ganges bei einem Schaltgetriebe entstehen.

Versuchsaufbau

Folgende Gleichung definiert für jeweils einen Quadranten (a)-(d) einer zweidimensionalen Eingabe verschiedene Dynamiken:

$$\mathrm{f}(x,y) = \begin{cases} \sqrt{x^2+y^2}\cdot\cos(x) & \text{für } x<0 \wedge y \geq 0 \quad \text{(a)} \\ x+y & \text{für } x \geq 0 \wedge y \geq 0 \quad \text{(b)} \\ 0 & \text{für } x<0 \wedge y<0 \quad \text{(c)} \\ \sqrt{x^2+y^2} & \text{für } x \geq 0 \wedge y<0 \quad \text{(d)} \end{cases} \tag{6.1}$$

Abbildung 6.5 zeigt die zu Formel (6.1) gehörige Funktionsfläche. Das ebenfalls gezeigte Beispiel einer Trajektorie wurde – wie dort beschrieben ist – zur Generierung einer Sequenz von $(\vec{x}_i, \vec{y}_i)$ Werten verwendet. Diese Sequenz wurde daraufhin mit LEMON gelernt.

Resultate

In Tabelle 6.1 ist eine Liste der von LEMON automatisch plazierten Modelle zu finden. Daraus ist ersichtlich, daß primär lineare Modelle verwendet wurden.

Das von LEMON gelernte Modell wurde mittels eines 1000×1000 Rasters abgefragt, das die von der Trajektorie überstrichene Eingabefläche abdeckt. Die absolute Differenz zur Originalfläche aus Gleichung (6.1) wurde eingefärbt, so daß Abweichungen von weniger als 0.5% bezogen auf den gesamten Wertebereich weiß erscheinen. Alle restlichen Fehler wurden als Verlauf von gelb für kleine Fehler bis rot für große Fehler dargestellt. In Abbildung 6.6 wurde zusätzlich noch die Lage der generierten Ellipsoide und die dadurch induzierte Voronoi-Parkettierung eingezeichnet.

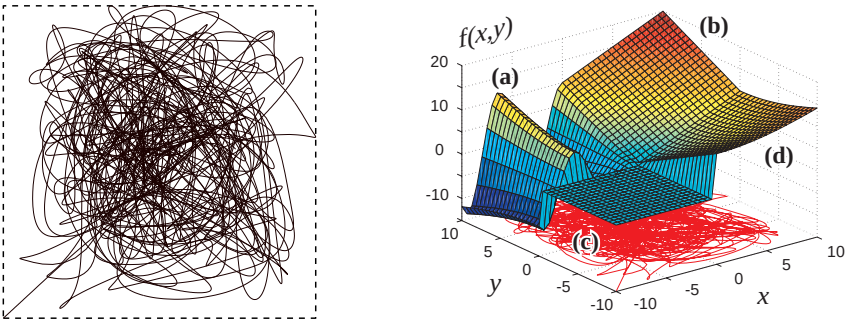


Abbildung 6.5: Hier ist ein Beispiel der verwendeten Trajektorien zu sehen, die zufällig gewählte Punkte mittels kubischer Splines verbinden. Die diskretisierten Trajektorienpunkte (x_i, y_i) wurden anschließend um den Funktionswert aus Formel (6.1) der rechts gezeigten Funktionsfläche zu $\{(x_i, y_i), \mathrm{f}(x_i, y_i)\}$ ergänzt. Daraus ergibt sich eine Beschreibung der Testfläche in Form einer kontinuierlichen Trajektorie.

Modelltyp	Parameterzahl	Anzahl
Multi Layer Perzeptron	25	1
RBF-Netz	10	1
RBF-Netz	37	1
Lineares Modell	3	22

Tabelle 6.1: Von LEMON mittels Crossvalidation gewählte lokale Modelle.

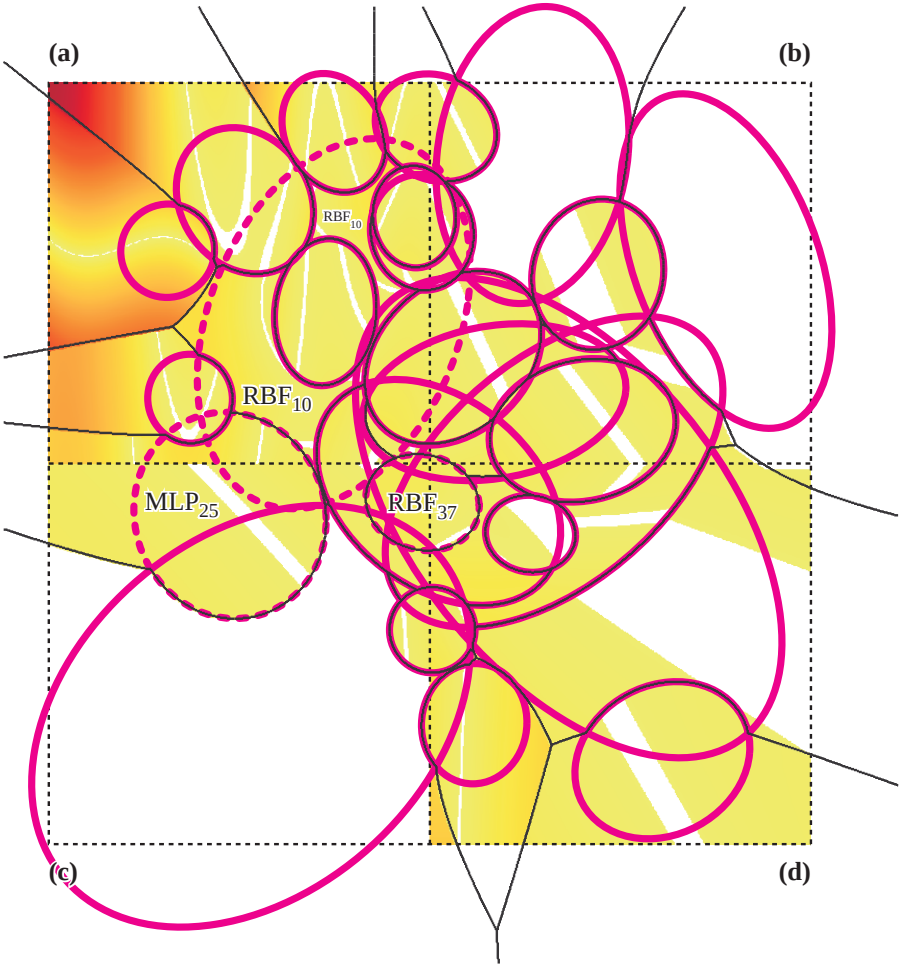


Abbildung 6.6: Hier ist ein Farbverlauf von gelb für kleine Fehler bis rot für große Fehler einer Modellierung der Funktion aus Gleichung (6.1) mit LEMON zu sehen. Weiße Bereiche gelten als exakt approximiert. Die zusätzlich magenta eingezeichneten Ellipsoide markieren den Bereich der präsentierten Lerndaten. Die Quadranten sind analog zu Formel (6.1) mit (a)-(d) bezeichnet. Deutlich sind die Übergänge zwischen den einzelnen Dynamiken erkennbar. Die roten Bereiche im linken oberen Teil befinden sich außerhalb der belernten Ellipsoide, wurden aber dennoch entsprechend der schwarz eingezeichneten Zugehörigkeit modelliert. Die in Tabelle 6.1 aufgeführten nichtlinearen Modelle sind im jeweiligen Einzugsbereich der (gestrichelten) Ellipsoide eingezeichnet. Der lineare Charakter der restlichen Modelle ist auch an den Übergängen weißer in gelbe Bereiche innerhalb der zugehörigen Ellipsoide erkennbar.

Beurteilung

An diesem Beispiel läßt sich sehr gut der Einsatz verschiedener Modelle durch LEMON studieren. So setzt es im Fall der Ebene und konstanten Fläche lineare Modelle ein, die sich optimal anpassen. Der Übergang zu den anderen Dynamiken erfolgt – mit kleinen Abweichungen – an den korrekten Stellen. Wie zu erwarten plaziert LEMON relativ viele – aber flächenmäßig kleine – Ellipsoide im Bereich des modulierten Cosinus, so daß auch diese Dynamik mit hinreichender Genauigkeit approximiert werden kann.

LEMON demonstriert seine Fähigkeit anhand einer On-Line gegebenen Trajektorie eine komplexe Dynamik zu modellieren. Die auftretenden größeren Fehler würden im realen Betrieb – anhand der Lage außerhalb der kartierten Region – erkannt und könnten so extra behandelt werden. Zudem treten wirklich relevante Fehler nur *außerhalb* der belernten Region auf, so daß in diesen Fällen vom nächstgelegenen Modell eine Extrapolation vorgenommen wird. Zu Bemerken ist noch die über den Lernverlauf relativ konstant bleibende Zahl der eingesetzen Submodelle. Befürchtungen einer „Modellexplosion“ haben sich als unberechtigt erwiesen, da durch den Löschvorgang nicht ausreichen konfidenter Modelle ein Gegengewicht zum Einfügen neuer (Sub-)Modelle geschaffen wurde.

6.2.2 Hysterese

Mit einer der unbeliebtesten nichtlinearen Effekte ist die Hysterese, die über einen theoretisch unendlichen Zustandsraum verfügt. Auf Seite 7 wurde schon darauf hingewiesen, daß zur Approximation vor allem Spezialverfahren zum Einsatz kommen, wie sie in [50, 60] beschrieben werden. Was aber, wenn ein versteckter bzw. unerkannter Hystereseeffekt vorliegt? In solchen Fällen wird meist ein Black-Box-Modell wie LEMON verwendet, um die Dynamik zu erfassen. Es ist also durchaus sinnvoll die generelle Tauglichkeit von LEMON auf diese Art von Dynamik hin zu testen.

Versuchsaufbau

Nachdem keine realen Hysterese-Daten zur Verfügung standen, wurde ein aus [60] abgeleiteter Generator für künstliche Hysterese-Daten verwendet. Dem LEMON-System wurde einmalig eine damit zufällig generierte Trajektorie der Länge 10000 präsentiert. Anschließend wurde das gelernte LEMON-Modell mit einer anderen zufällig generierten Testtrajektorie der Länge 10000 abgefragt und mit dem Ergebnis des Generators verglichen. Die gewählte Parametrierung hatte zum Ziel, relativ wenige lokale Modelle zu verwenden, um ein einfaches „Abspeichern“ der Datenpunkte ohne Generalisierung zu verhindern. Zusätzlich zur aktuellen wurden auf die letzte und vorletzte Eingabe der Hysterese präsentiert. Die zugehörigen Zeitpunkte werden im weiteren mit t_0, t_{-1} und t_{-2} bezeichnet. Es wurden zwei verschiedene LEMON-Modelle belernt. Das erste hatte die in Tabelle 6.2 aufgeführten Modell-Prototypen zur Verfügung, das zweite nur RBF-Netze unterschiedlicher Komplexität.

Resultate

Wie Tabelle 6.2 zu entnehmen ist, hat das erste durch LEMON erstellte Modell 24 lokale Modelle plaziert. Ein Großteil davon ist linear, nur an wenigen Positionen wurden nichtlineare Modelle positioniert.

Modelltyp	Parameterzahl	Anzahl
Multi Layer Perzeptron	16	1
Multi Layer Perzeptron	31	1
RBF-Netz	13	2
RBF-Netz	25	0
RBF-Netz	49	0
Lineares Modell	4	20

Tabelle 6.2: Von LEMON mittels Crossvalidation gewählte lokale Modelle.

In Abbildung 6.7 ist jeweils ein Ausschnitt der Vorhersage der beiden LEMON-Hysteresemodelle der gelernten Trainingstrajektorie gezeigt. Wie man auch Tabelle 6.3 entnehmen kann, ist das nur mit RBF-Netzen ausgestattete LEMON-Modell um ca. 1% schlechter, als das mit einem Modell-Mix versehene. Dies ist auf die in der Regel lokal lineare Natur der Hysterese zurückzuführen. Offensichtlich wurde diese auch erkannt, da vom ersten LEMON-Modell fast nur lineare Modelle eingesetzt werden.

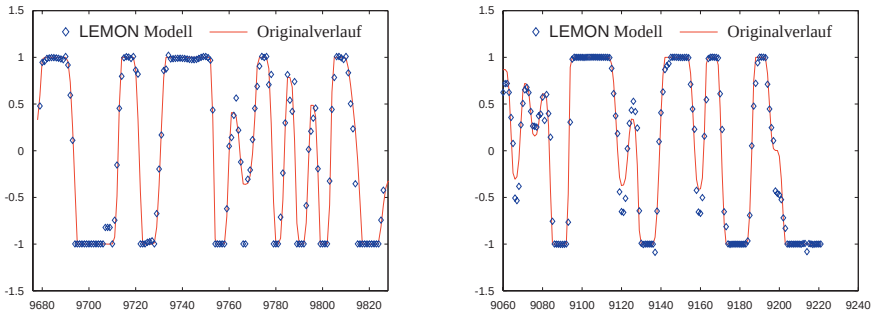


Abbildung 6.7: Ausschnitte aus den Vorhersagen der LEMON-Hysteresemodelle mit der Trainingstrajektorie als Eingabe. Links ist das Modell mit verschiedenen lokalen Modell-Prototypen gezeigt. Rechts das Modell mit RBF-Netzen. Es wurde jeweils der beste Ausschnitt gewählt.

Wie bereits in Abbildung 6.7, so ist auch in Abbildung 6.8 kein wesentlicher Unterschied zwischen der Performanz der beiden Modelle zu erkennen. Dies ist vor allem auf die Wahl der jeweils besten Ausschnitte bezüglich des absoluten Fehlers zurückzuführen. Im Gegensatz zur vorherigen Abbildung handelt es sich hier um eine Vorhersage mit einer bisher unbekannte Testtrajektorie.

Betrachtet man die in Abbildung 6.9 gezeigten *schlechtesten* Ausschnitte bezüglich des absoluten Fehlers, so erkennt man deutliche Unterschiede. Findet sich in der linken Vorhersage die Originalkurve noch wieder, so weicht dies in der rechten Vorhersage einem teils chaotischen Fehler. Dies ist durch den lokalen Charakter der RBF-Netze zu erklären, die hier nicht ausreichend generalisieren um mit den in linken Teil der Abbildung verwendeten linearen Modellen konkurrieren zu können.

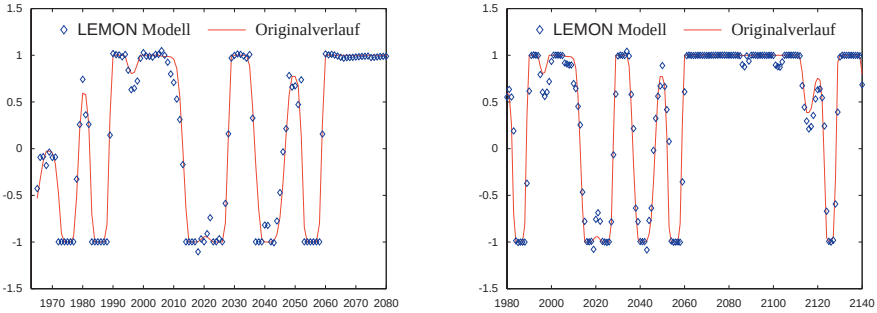


Abbildung 6.8: Ausschnitte der Vorhersage der LEMON-Hysteresemodelle mit einer Teststrajektorie als Eingabe. Links ist das Modell mit verschiedenen lokalen Modell-Prototypen gezeigt. Rechts das Modell mit RBF-Netzen. Es wurde jeweils der beste Ausschnitt gewählt.

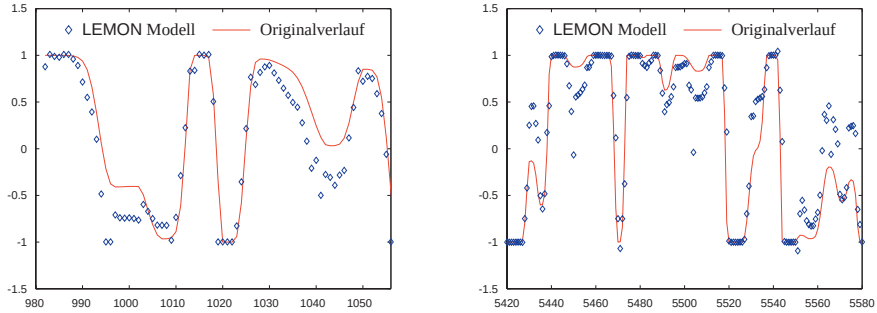


Abbildung 6.9: Ausschnitte der Vorhersage der LEMON-Hysteresemodelle mit einer Teststrajektorie als Eingabe. Links ist das Modell mit verschiedenen lokalen Modell-Prototypen gezeigt. Rechts das Modell mit RBF-Netzen. Es wurde jeweils der schlechteste Ausschnitt gewählt.

In Abbildung 6.10 ist die Zustandsraumzerlegung durch das LEMON-Hysteresemodell mit 24 lokalen Modellen zu sehen. Anhand der Projektionen (a)-(c) erkennt man, daß die eigentliche Information offensichtlich durch die Kombination der t_0- und t_{-2}-Achse kodiert wurde, da Ellipsoide in (c) stärker gestreut sind als in anderen Achskombinationen. Bemerkenswert ist auch die Positionierung der nichtlinearen Modelle am Übergang in den Sättigungsbereich der Hysterese. Ein lineares Modell wäre fehl am Platz, da aus verschiedensten Richtungen ein Eintritt in die Sättigung erfolgen kann. Die 3D-Ansicht in (d) zeigt dann noch die komplette Abdeckung des Zustandsraumes, wie sie aus der Trainingstrajektorie von LEMON extrahiert wird.

Mittelt man den absoluten Vorhersage-Fehler aller Vorhersagen für verschiedene LEMON-Parametrierungen, so erhält man die in Tabelle 6.3 aufgeführten Trainings- und Testfehler. Die Parameter wurden so variiert, daß nach *einem* Durchlauf der Trainingstrajektorie verschiedene Anzahlen lokaler Modelle resultierten. Zusätzlich dazu ist der Anteil linearer Modelle jeweils in Klammern angegeben.

Gesamtzahl			**Fehler**	
Modelle	**(Linear)**	**Parameter**	**Training**	**Test**
18	– RBF –	402	6.8%	6.9%
24	(20)	153	5.8%	6.1%
112	(72)	1186	5.2%	5.4%
234	(156)	2328	4.1%	4.2%
517	(360)	4921	4.5%	4.9%
828	(634)	6882	4.8%	5.0%
RBF-Netzwerk		2321	4.4%	4.7%

Tabelle 6.3: Absolute Vorhersage-Fehler der Hysteresemodelle von LEMON nach jeweils einmaligem Durchlauf der Trainingstrajektorie. Das Ergebnis eines RBF-Netzwerkes nach 1000 Durchläufen findet sich am Ende der Tabelle.

Daß sowohl der Trainings-, als auch der Testfehler gleichermaßen fallen bzw. steigen zeigt die Generalisierungsfähigkeit von LEMON, da der Testfehler ansonsten willkürlich steigen bzw. fallen müßte. Die optimale Modellzahl lag bei 234, mit mehr lokalen Modellen steigt der absolute Fehler wieder an. Dies ist auf „Verschattungseffekte“ zurückzuführen. Dabei wird durch nachträgliches Einfügen eines neuen lokalen Modells die Zugehörigkeit bereits klassifizierter Datenpunkte modifiziert. Da kein zweiter Durchlauf erfolgt, kann dies nicht korrigiert werden.

Zum Vergleich wurde ein RBF-Netzwerk mit 2321 Parametern – dies entspricht der Komplexität des besten LEMON-Modells aus Tabelle 6.3 – zufällig initialisiert und anschließend mit 1000 Durchläufen durch die Trainingstrajektorie belernt. Der Lernvorgang des LEMON-Modells dauerte für *einen* Durchlauf auf einer SUN Ultra-60 Workstation mit zwei 296 MHz Prozessoren 2 Minuten. Der Lernvorgang des RBF-Netzes erforderte auf demselben Rechner mehrere Stunden. Dennoch liegt die Performanz von LEMON über der des RBF-Netzes.

Beurteilung

Dieses Beispiel zeigt, daß LEMON in Grenzen auch Hystereseeffekte nachbilden kann. Insbesondere wenn diese nicht den bestimmenden Anteil der zu lernenden Dynamik darstellen, läßt sich diese gut approximieren. Der Vergleich mit einfachen Ansätzen wie etwa RBF-Netzen, hat

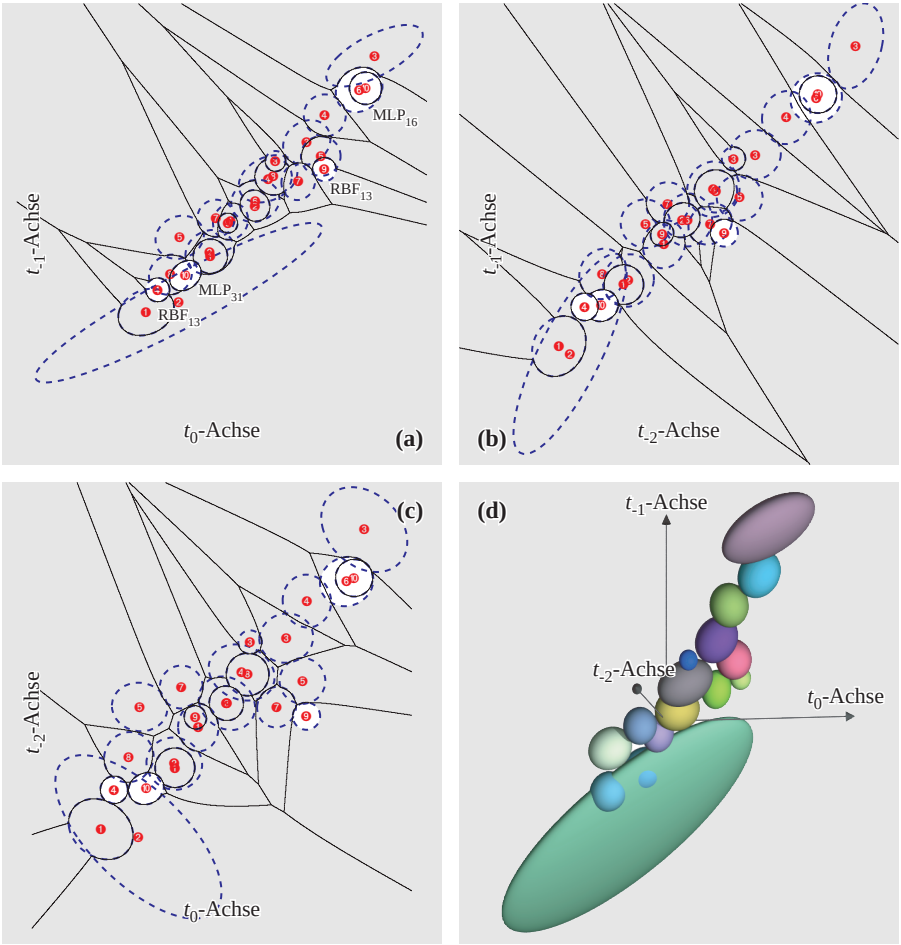


Abbildung 6.10: Zustandsraumzerlegung einer Hysterese durch LEMON. Nichtlineare Modelle sind in (a)-(c) weiß markiert und in (a) zusätzlich beschriftet. Im einzelnen sind zu sehen: **(a)** Projektion auf die t_0- und t_{-1}-Achse. **(b)** Projektion auf die t_{-1}- und t_{-2}-Achse. **(c)** Projektion auf die t_0- und t_{-2}-Achse **(d)** 3D-Ansicht aller Achsen.

eine deutliche Überlegenheit von LEMON gezeigt. Diese ist nicht nur in der On-Line-Fähigkeit, sondern auch in der Lerngeschwindigkeit und Approximationsgüte erkennbar.

Die stark schwankende Approximationsgüte bei Nutzung unterschiedlicher Prototypmodelle zeigt, daß LEMON den lokal linearen Charakter der Hysterese korrekt erkannt hat und primär lineare Modelle einsetzt. Dies führt zu einer sehr guten Generalisierung, wie an dem geringen Unterschied zwischen Trainings- und Testfehler abgelesen werden kann.

6.2.3 Prüfstandssimulation

Bei der folgenden Simulation handelt es sich um einen Prüfstandsaufbau zum Test eines Ottomotors. Ziel dabei ist es einem vorgegebenen Geschwindigkeitsprofil bestmöglich zu folgen. Ein Regler kann dazu die Gaspedalstellung $\alpha \in [0, 90°]$ beeinflussen und somit die Fahrzeuggeschwindigkeit regeln. Die Rolle des Fahrzeugs wird dabei von einem Fahrzeugmodell, das eine E-Maschine ansteuert übernommen. Es soll gezeigt werden, daß mit Hilfe eines inversen LEMON-Prozeßmodells eine Verringerung der Regelgrößenabweichung vom Sollverlauf erreicht werden kann.

Versuchsaufbau

Da kein realer Prüfstand mit ausreichender Anbindung zum Test verschiedener Modelle zur Verfügung stand, wurde die in Abbildung 6.12 gezeigte MATLAB-Simulation entwickelt. Sie besteht aus den drei Teilen Prüfstand, Fahrzeugmodell und Regler.

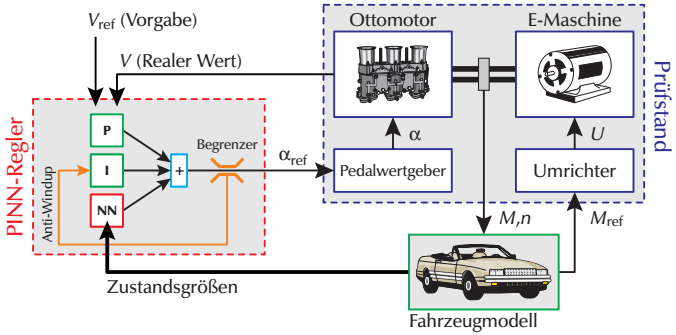


Abbildung 6.11: Aufbau der MATLAB-Prüfstandsimulation mit PINN-Regler.

Bein Prüfstand handelt es sich um die Simulation eines realen Motorenprüfstandes. Das Fahrzeugmodell liefert anhand der aktuellen Drehzahl und des Motordrehmoments eine Drehmomentvorgabe M_{ref}, die mittels eines Umrichters als Spannung U an die E-Maschine weitergegeben wird. Dadurch wird das in einem realen Fahrzeug auf den Motor wirkende Drehmoment simuliert.

Um eine modellbasierte Regelung zu ermöglichen, wurde der PI-Regler so erweitert, daß mit Hilfe eines inversen Prozeßmodells (siehe auch Seite 12) eine höhere Reaktionsgeschwindigkeit als mit dem ursprünglichen PI-Regler erreicht werden kann. Das inverse Prozeßmodell ist in Abbildung 6.12 innerhalb des Reglers zu finden und mit NN bezeichnet. Die zum Training und zur Vorhersage notwendigen Zustandsgrößen werden dabei vom Fahrzeugmodell geliefert.

Der resultierende PINN-Regler ist zusammen mit der Prüfstandsimulation bereits in [106] vorgestellt worden.

Resultate

Abbildung 6.12 zeigt zwei Ausschnitte einer Prüfstandsimulation mit PINN-Regler und einem LEMON-Modell. Gezeigt ist jeweils die Winkelgeschwindigkeit der Reifen ω, welche proportional zur Geschwindigkeit v ist. Im linken Teil ist zu sehen, wie der Einschwingvorgang auf die Sollvorgabe durch Einsatz des PINN-Reglers beschleunigt und ein neuerliches Überschwingen vermieden wird. Obwohl modellbedingt das erste Überschwingen stärker ausfällt, findet die Annäherung an die Sollkurve schneller statt. Auch im rechten Teil der Abbildung wird eine Verbesserung erzielt, diesmal nicht bei einem Beschleunigungs-, sondern bei einem Bremsvorgang.

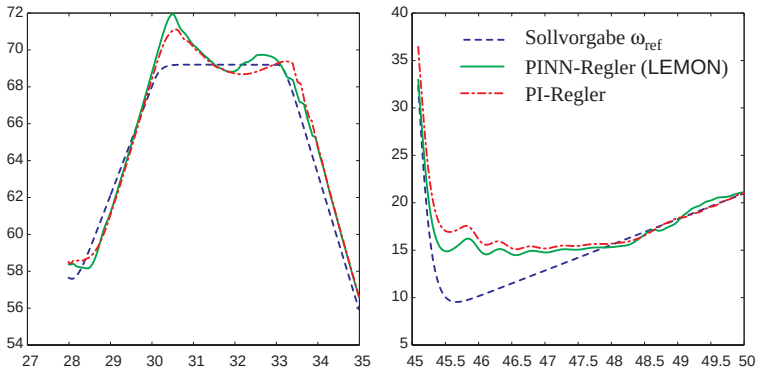


Abbildung 6.12: Ergebnis einer Prüfstandsimulation mit PINN-Regler und LEMON.

Tabelle 6.4 zeigt die prozentuale Verbesserung der Performanz durch Einsatz des PINN- anstelle des PI-Reglers mit verschiedenen Modellen. Gemessen wurde jeweils der Fehlerquadrat-Abstand der gesamten gefahrenen Geschwindigkeitstrajektorie von der Sollvorgabe. Es wurden zwei unterschiedliche Simulationen durchgeführt: Die erste mit einem Mittelwertfahrzeugmodell, die zweite mit einer zusätzlichen Oberschwingung auf dem Momentenverlauf.

Modell	Verbesserung	
	Simulation 1	Simulation 2
ATNN	2.9%	4.5%
DRBF	1.3%	7.1%
LEMON	9.0%	10.9%

Tabelle 6.4: Prozentuale Verbesserung der Performanz durch Einsatz des PINN-Reglers.

Beurteilung

Wie aus Tabelle 6.4 zu entnehmen ist, schlägt LEMON in beiden Simulation die Neuronalen Netze ATNN und DRBF deutlich. Ein Vorteil beim Einsatz von LEMON ist, daß es unbelernte

Bereiche erkennt und deaktiviert werden kann. Der PINN-Regler wird somit als normaler PI-Regler weiterbetrieben. Die einfache Strategie der additiven Aufschaltung auf die PI-Stellgröße sollte allerdings einer „intelligenten“ Aufschaltung weichen. Damit ist gemeint, daß Modelle nur in Bereichen aktiviert werden sollten, in denen der PI-Regler überfordert ist. Dies ist allerdings Aufgabe des Reglers und nicht des Modells. Dennoch sollte auch die prinzipielle Tauglichkeit des eingesetzten Modells für die jeweilige Reglerstruktur bedacht werden. (Wie etwa Fähigkeit zum On-Line-Training, Erkennen ungelernter Bereiche, usw.)

6.2.4 Vergleich verschiedener Verfahren am Heizungsmodell

Im folgenden Abschnitt findet sich ein Vergleich von LEMON mit ATNN- und DRBF-Netzen an einem Datensatz, der mit dem im Anhang A zu findenden Heizungsmodell generiert wurde. Der Datensatz ist für verschiedene statische und dynamische Modelle bereits in [104] untersucht worden und ist dort näher erläutert.

Versuchsaufbau

Als Eingaben erhält LEMON den Verlauf der Kesseltemperatur T_K, die Heizkörpertemperatur T_H und die Außentemperatur T_A. Ziel ist es, die Innentemperatur T_I vorherzusagen. Aus numerischen Gründen erfolgte eine reversible Normierung der Ein-/Ausgabedaten auf den Wertebereich $[-1, 1]$. Um einen Vergleich mit den in [104] gemachten Versuchen zu ermöglichen, erfolgte die Vorhersage mit demselben Datensatz, der auch zum Lernen angeboten wurde.

Resultate

Das Ergebnis der Modellierung in Abbildung 6.13 wurde bereits nach 10 Durchläufen erzielt. Dazu positionierte LEMON insgesamt nur 10 *lineare* Modelle mit insgesamt 40 Parametern, was auf eine gute Generalisierung hinweist. Die Vorhersage der Innentemperatur durch LEMON erfolgt fast korrekt. Die Zustandsraumzerlegung durch LEMON zeigt, daß die drei Achsen T_H, T_K und T_A voneinander unabhängig sind und bestätigt, daß gut positionierte lokale lineare Modelle ausreichen, um ein gute Vorhersage und Generalisierung zu erreichen.

Zum Vergleich ist in Abbildung 6.14 die entsprechende Vorhersage eines ATNN- und DRBF-Netzes gezeigt. Beide Netze haben Probleme den Innentemperaturverlauf ähnlich gut wie LEMON zu approximieren. Eine nähere Analyse zeigt, daß Probleme genau bei den von LEMON erkannten Dynamikwechseln auftreten.

Beurteilung

Bei diesem Beispiel ist einerseits die sehr gute Vorhersage der Innenraumtemperatur interessant, andererseits kann aus der Zerlegung des Zustandsraumes und den zugeordneten Modellen auch Aufschluß über das zugrunde liegende Modellierungsproblem gewonnen werden.

Vor der Modellierung des Datensatzes durch LEMON war nicht ersichtlich, daß eine einfache, lokal lineare Approximation ausreicht, um den Innentemperaturverlauf exakt vorherzusagen. Nach der Visualisierung des Zustandsraumes wird sofort deutlich, daß die drei Eingabegrößen unabhängig sind. Somit erscheint die von LEMON gewählte lokal lineare Approximation sinnvoll.

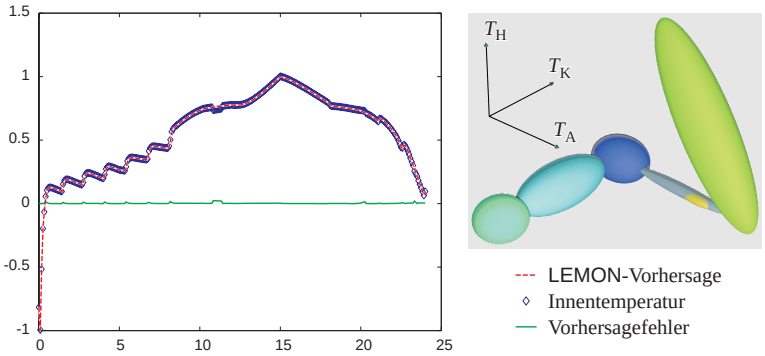


Abbildung 6.13: Links ist die Modellierung des Heizungsdatensatzes mit Hilfe von LEMON gezeigt. Die rot gestrichelte Linie markiert den Verlauf des Originalsystems, die blauen Rauten zeigen die Vorhersagen von LEMON. Da beide Kurven quasi exakt aufeinander liegen, ist zusätzlich grün durchgezogen der immer nahe Null liegende absolute Fehler angetragen. Rechts ist die zugehörige Zustandsraumzerlegung gezeigt.

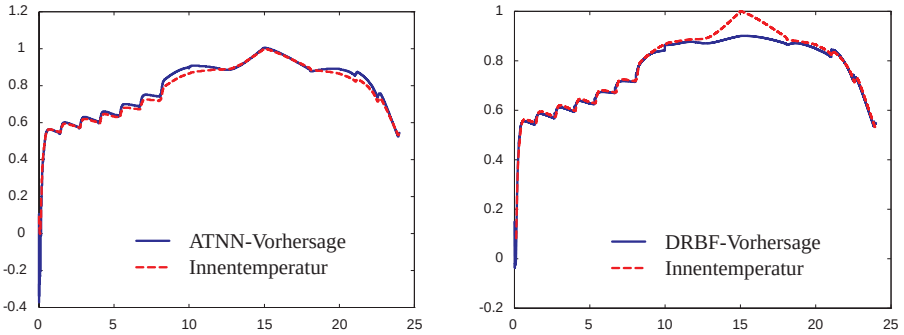


Abbildung 6.14: Die oben gezeigten Grafiken stammen ursprünglich aus [104]. Aus Platzgründen wurde die Beschriftung entfernt. Wie in Abbildung 6.13 zeigt die blau durchgezogene Linie den Modellverlauf und die rot gestrichelte Linie das Originalsystem.

Nachdem durch die in diesem Kapitel vorgestellten Simulationen die Fähigkeiten des entwickelten Filter- und Modellbildungsverfahrens demonstriert wurden, folgt nun eine Zusammenfassung der erreichten Ziele.

Kapitel 7

Schlußbetrachtung

Aufbauend auf den in den Simulationen erzielten Ergebnissen wird zuerst eine Zusammenfassung der erreichten Ziele aus Abschnitt 1.1 gegeben. Anschließend wird LEMON von ähnlichen Ansätzen abgegrenzt. Ein Ausblick auf mögliche weitere Forschungsziele schließt dieses Kapitel ab.

7.1 Zusammenfassung

In dieser Arbeit wurde ein neues Filter- und Modellbildungsverfahren zum Einsatz in der Prozeßmodellierung entworfen. Im folgenden werden die erzielten Resultate zusammenfassend dargestellt.

Regressionsfilter mit Sprungerkennung

Der im Abschnitt 4.1.6 entworfene Regressionsfilter mit Sprungerkennung mittels Varianztest kann verrauschte Daten mit Ausreißern und Sprüngen On-Line behandeln. Dies wurde mit umfangreichen Tests, auch im Vergleich zu klassischen Filtern, gezeigt. Darüberhinaus kann durch die neue *explizite* Darstellung der Pseudoinversen für äquidistante Regressionsprobleme die Einsatzfähigkeit auch in Rechnern geringer Kapazität garantiert werden.

LEMON

Das On-Line-Verfahren zur lokalen Modellbildung LEMON aus Kapitel 5 erfüllt eine ganze Reihe von Anforderungen. Im weiteren werden diese anhand der in der Einleitung erwähnten Ziele beschrieben:

On-Line-Fähigkeit. Da ein Ansatz mit lokaler Modellierung gewählt wurde, kann auch im laufenden Betrieb neues Wissen erlernt werden. Dabei garantiert die strikte Trennung der lokalen Modelle, daß Lernen nur lokal begrenzte Auswirkungen hat und somit bereits vorhandene Information erhalten bleibt. Simulationen, wie etwa das Erlernen einer Hysterese anhand einer zufälligen Trajektorie (Abschnitt 6.2.2), zeigen die praktische Tauglichkeit des entwickelten On-Line-Verfahrens.

Einbringen von Vorwissen. Durch die in Abschnitt 5.3.2 vorgestellte Modellauswahlkomponente kann mit Hilfe von Fuzzy-Regeln intuitiv prozeßspezifisches Wissen in LEMON eingebracht werden. Somit ist die Forderung nach Einbindung von Expertenwissen erfüllt.

Modellierung nichtlinearer Systeme. Da es in LEMON möglich ist, für jede Raumregion einen unterschiedlichen Modelltyp einzusetzen, ergibt sich automatisch die Fähigkeit zur Modellierung verschiedener nichtlinearer Dynamiken. Auch Sprünge sind durch Übergang von einer Region in die nächste modellierbar.

Generalisierung. Der fast ausschließliche Einsatz linearer Modelle durch LEMON in den Simulationen des letzten Kapitels zeigt die hohe Generalisierungfähigkeit des Ansatzes. Offensichtlich wird die lokale Modellkomplexität an die Prozeßkomplexität bzw. Komplexität der verschiedenen Dynamiken angepaßt. Auch die Vorgabe entsprechender Modellauswahlregeln kann sich dabei positiv auswirken. Durch die Expansion des Einzugsbereichs lokaler Modelle in Richtung guter Vorhersagen wird ebenfalls die lokale Generalisierung unterstützt.

Interpretierbar. Zahlreiche Beispiele aus dem vorherigen Kapitel zeigen, daß eine Visualisierung der Zustandsraumkartierung Aufschluß über den zu modellierenden Prozeß gibt. Nutzt man darüberhinaus die Möglichkeit, die Zuordnung der Modelltypen zu den einzelnen Regionen zu untersuchen, so erhält man einen umfassenden Einblick in die Dynamik des approximierten Prozesses.

Abschließend sei erwähnt, daß zum Erreichen der oben aufgeführten Ziele eine Reihe von grundlegenden Neuentwicklungen notwendig waren. So wurde eine ellipsoide Metrik entworfen, die bei Ellipsoiden Karten eine korrekte Abstandsberechnung zuläßt. Durch den Einsatz einer Fehlerabschätzung konnte die Berechnungskomplexität dieser Metrik wesentlich verringert werden, so daß ein Einsatz auf derzeit verfügbaren Rechnern ermöglicht wurde. Die Kopplung eines On-Line-Clusterverfahren mit der Performanz der zugehörigen lokalen Modelle ist ebenfalls eine Neuerung.

7.2 Abgrenzung

Nach dieser Zusammenfassung soll LEMON nun noch im Umfeld ähnlicher Architekturen eingeordnet werden. Im wesentlichen existieren drei Ansätze, die Ähnlichkeiten zu LEMON aufweisen. Dies sind Multiagentensysteme, „Mixture of Experts“-Architekturen und intelligente hybride Systeme.

7.2.1 Multiagentensysteme

Der Bereich der Multiagentensysteme (MAS) hat in jüngerer Zeit rasch wachsende Aufmerksamkeit erfahren (siehe z.B. [28, 42, 72, 107] für aktuelle Publikationen). Generell wird unter einem MAS ein System verstanden, welches aus mehreren interagierenden Agenten besteht, die eine Menge von (gemeinsamen oder verschiedenen) Zielen verfolgen. Zwar gibt es nach wie vor keine einheitliche Definition von „Agent“ (ein Überblick über verschiedene Sichtweisen dieses Begriffes findet sich beispielsweise in [30]), generell jedoch zeichnet sich eine breite Zustimmung zu folgender, im Wesentlichen auf den Arbeiten von Wooldridge und Jennings [113] basierenden Charakterisierung ab: Ein Agent ist ein Objekt, welches in der Verfolgung seiner (vom Benutzer vorgegebenen) Ziele als flexibel, interaktiv und autonom bezeichnet werden kann:

Flexibilität. Die angemessene Reaktion des eigenen Verhaltens an sich ändernde Anforderungen und Umgebungen.

Autonomie. Die Fähigkeit – weitgehend – unabhängige Entscheidungen zu treffen, ohne Rücksprache mit dem Benutzer oder dem übergeordneten System zu halten.

Interaktivität. Die Kommunikation mit anderen Agenten bzw. Menschen, im Sinne „sozialer“ Interaktion. Dies kann zum Beispiel ein Erfahrungsaustausch, das Aushandeln einer gemeinsamen, „globalen“ Strategie oder der Abschluß eines Vertrages sein.

In Hinblick auf eine MAS-orientierte Sicht von LEMON liegt es nahe, jeder Modellregion einen eigenen Agenten zuzuordnen. Diese Sicht ist bezüglich Autonomie und Flexibilität durchaus plausibel, da die Erweiterung eines lokalen Einzugsbereichs *autonom* möglich ist und die lokal eingesetzten Modelle sich schleichenden Veränderungen *flexibel* anpassen können. Bezüglich Interaktivität ist diese Sichtweise aber problematisch. In der bei LEMON gewählten Strategie der lokalen Modellbildung wurde aus Gründen der Zeitkomplexität des sonst entstehenden Koordinations- und Kommunikationsaufwands ganz bewußt auf Interaktion zwischen den einzelnen Regionen verzichtet. Es handelt sich bei den einzelnen Regionen also eher um Spezialisten, da die einmal lokal festgelegte Modellstruktur nur adaptiert oder gelöscht werden kann. Der Analogie zwischen LEMON und MAS sind somit Grenzen gesetzt.

7.2.2 „Mixture of Experts“-Architekturen

Von Jacobs, Jordan, Nowlan, und Hinton [45] wurde 1991 eine Architektur mit dem Namen „Mixture of Experts“ (MOE) vorgestellt. Diese besteht aus einer festen Menge von n Modellen $\mathrm{f}_i(\vec{u})$, die Experten genannt werden. Der Beitrag jedes Experten zur Gesamtaussage des Systems wird über Gewichtungsfunktionen $\mathrm{g}_i(\vec{u})$ gesteuert, die auch „Gates“ genannt werden.

Als Ausgabe des Systems kann nun einerseits $y = \sum_{i=1}^{n} \mathrm{g}_i(\vec{u})\mathrm{f}_i(\vec{u})$ gewählt werden, womit jeder Experte *immer* einen Beitrag leistet. Andererseits ist auch ein hartes Umschalten zwischen den Experten möglich, was zur Ausgabe $y = \mathrm{f}_k(\vec{u})$ mit $\forall i \neq k : \mathrm{g}_i(\vec{u}) \leq \mathrm{g}_k(\vec{u})$ führt. Um eine Interpretation als Wahrscheinlichkeit zu ermöglichen, wird meist die erste Form gewählt und die Gewichtungsfunktion mit Hilfe der sogenannten Soft-Max-Funktion realisiert. Diese erzwingt positive Gewichtungsfunktionen und garantiert deren Gesamtsumme von Eins. Verfügt man über beliebige Gewichtungskriterien $z_i(\vec{u})$, so erhält man die Gewichtungsfunktionen somit als $\mathrm{g}_i(\vec{u}) = \frac{e^{z_i(\vec{u})}}{\sum_{j=1}^{n} e^{z_j(\vec{u})}}$.

In den letzten Jahren wurden eine Reihe von Erweiterungen bzw. Variationen dieser Architektur vorgestellt, unter anderem [47, 114, 62, 108]. Dabei handelt es sich einerseits um Modifikationen des Gewichtungsmechanismus, andererseits um Variationen der verwendeten Experten.

Vergleicht man die MOE-Architekturen mit LEMON, so erscheinen diese nur auf den ersten Blick identisch: Beide besitzen Experten bzw. Modelle, die sich auf bestimmte Bereiche spezialisieren. Ist die Zahl und Art bei MOE von vornherein fest, so verfügt LEMON über einen flexiblen Mechanismus zur Selektion und Plazierung bzw. Löschung dieser. Bei MOE findet bevorzugt ein sanftes Umschalten zwischen den Experten statt. LEMON kennt nur einen harten Wechsel zwischen den Modellen. Verschiedene Autoren [62, 54] berichten von besseren Ergebnissen beim Lernen nichtlinearer Probleme, wenn hart umgeschaltet wird. Bisher bekannte MOE-Architekturen nutzen zur Gewichtung der Ausgabe *entweder* die Eingabe *oder* die Leistung der einzelnen Experten bei den letzten Vorhersagen. Der in LEMON implementierte Algorithmus zur Expansion der Ellipsoide (=Experteneinzugsgebiete) berücksichtigt *sowohl* die Eingabe *als auch* die Leistung. Auch das bei LEMON mögliche Einbringen von Vorwissen ist bei MOE nicht vorgesehen. Es läßt sich also allenfalls eine entfernte Verwandtschaft zwischen LEMON und MOE feststellen.

7.2.3 Intelligente hybride Systeme

Aus der Erkenntnis heraus, daß viele „reale“ Probleme nicht mit einer einzigen Methodik zu lösen sind, wurden in den letzten Jahren verstärkt sogenannte „hybride“ Systeme eingesetzt. Dabei handelt es sich um die Kombinationen verschiedenster Verfahren, die jeweils unterschiedliche Stärken und Schwächen besitzen. Durch Wahl einer geeigneten Gesamtstruktur können die jeweiligen Verfahren so gekoppelt werden, daß sie ihre Stärken optimal einsetzen.

Folgt man der Definition von Goonatilake und Khebbal [35], die intelligente hybride Systeme in die drei Klassen „Funktionserweiternd“, „Kommunizierend“, und „Polymorph“ aufteilen, so läßt sich LEMON eindeutig der zweiten Klasse zuordnen. Wichtig hierbei ist, daß die Kommunikation auch *indirekt*, also über einen globalen Koordinator, möglich ist und sich somit von der in Multiagentensystemen genannten Kommunikation unterscheidet. Als wesentliche Eigenschaften eines intelligenten Systems werden die folgenden Punkte betrachtet:

Automatische Wissenserfassung. Die Fähigkeit selbständig neues Wissen zu erfassen, einzuordnen und später wieder abzurufen.

Stabilität. Bei Änderung der bisher erfaßten Umwelt reagieren viele Systeme sensibel und liefern bereits bei kleinen Abweichungen inkonsistente Ergebnisse, sind also instabil.

Abstrakte und konkrete Wissensmodellierung. Die reale Umwelt läßt sich einerseits durch abstrakte Regeln, andererseits durch konkrete Vorgänge, wie sie etwa durch Mustererkennung oder Funktionsapproximation erfaßt werden, charakterisieren.

Interpretierbarkeit. Gemeint ist die Möglichkeit einzelne Schritte der Wissensmodellierung nachvollziehen zu können. Auch das Gewinnen neuer Erkenntnisse über das zu lösende Problem ist Teil der Interpretierbarkeit.

Alle Punkte sind – in unterschiedlichem Maße – für LEMON erfüllt. Somit läßt es sich als ein „intelligentes hybrides System“ bezeichnen. Der nun folgende Ausblick gibt Anregungen, wie dieses System noch erweitert und für ein breiteres Anwendungsfeld nutzbar gemacht werden kann.

7.3 Ausblick

Ausgehend von den in dieser Arbeit erzielten Resultaten erweisen sich die folgenden weiteren Forschungsrichtungen als besonders naheliegend.

Verteilte Realisierung von LEMON

Zur Zeitoptimierung können die einzelnen Regionen als nebenläufige Prozesse realisiert werden. Gestattet man (in engem Rahmen) Kommunikation zwischen den Prozessen, dann ist hier auch die Anwendbarkeit von MAS-Technologie (siehe oben) neu zu bewerten. Denkbar wäre dann beispielsweise auch, eine „Aufweichung“ der Grenzbereiche zwischen den Regionen vorzunehmen, so daß die „regionalen Agenten“ ihre Zuständigkeiten aushandeln und gemeinsam optimieren. Dabei wird insbesondere zu untersuchen sein, in welchem Verhältnis die erzielte qualitative Verbesserung und der zusätzliche zeitliche Aufwand für Interaktion stehen. Auch der Austausch von Modellen zwischen den Regionen ist dann möglich, so daß vom Wissen benachbarter Regionen profitiert werden kann.

Integration von Reglern und LEMON

Zur Verbesserung der Performanz bei Regelungsaufgaben wäre der Entwurf eines Reglers denkbar, der die Konfidenzinformation von LEMON nutzt. Auch das Erkennen unbelernter Regionen wäre so möglich. Anhand dieser zusätzlichen Informationen ist der Regler dann in der Lage, zwischen modellbasiertem und klassischem Betrieb zu wechseln.

Eine andere Möglichkeit ist es, anstelle von lokalen Modellen lokale Regler zu verwenden. Damit wäre das ellipsoide On-Line-Clusterverfahren von LEMON nutzbar, um anhand der Reglerperformanz „regelbare“ Bereiche zuzuteilen.

Theoretische Untersuchung verschiedener Konfidenzmaße

Die in LEMON verwendete Heuristik zur Konfidenzschätzung könnte durch eine wahrscheinlichkeitsbasierte Variante ersetzt werden. Dies hätte den Vorteil, daß die darauf basierenden Entscheidungsalgorithmen, wie etwa das Löschen einer Region, in ihrer genauen Wirkungsweise mathematisch analysiert und entsprechend modifiziert werden können.

Für den Einsatz eines solchen Konfidenzmaßes sind genaue theoretische Untersuchungen der dann anwendbaren lokalen Modelle notwendig. Auch ist zu überprüfen, ob die dadurch zusätzlich auftretende Berechnungskomplexität mit einer Verbesserung der Modellierungsleistung einhergeht.

Untersuchungen zum Einsatz von Expertenwissen

Die Kombination verschiedener Prototypmodelle mit prozeßspezifischen Auswahlkriterien trägt entscheidend zur Performanz von LEMON bei. Die Auswirkungen spezieller Kombinationen sollte somit untersucht werden. Auch der Einsatz neuronaler Netze zur Modellselektion stellt in diesem Zusammenhang ein interessantes Thema dar.

Verwendung von LEMON zur Datenanalyse

Durch seine gute Interpretierbarkeit ist LEMON nicht auf die On-Line-Modellbildung beschränkt, sondern bietet sich auch als Werkzeug zur Datenanalyse an. Eine Erweiterung der Visualisierungsmöglichkeiten und der Entwurf einer geeigneten Benutzerschnittstelle erscheint hier sinnvoll.

Die in dieser Arbeit erzielten Ergebnisse bilden für diese Themen eine solide Grundlage zur weiteren Forschung.

Anhang A

Simulation einer Gebäudeheizung

Die im folgenden beschriebene Simulation wurde bereits in [94] vorgestellt, und eignet sich insbesondere zum Test von Verfahren zur Totzeiterkennung. Sie zeichnet sich durch einen modularen Aufbau aus, so daß verschiedene Raumkonstellationen einfach erstellt werden können. Bei der Beschreibung wurden für physikalische Größen jeweils üblicher Standardwerte mit angegeben. Zusätzlich befinden sich am Ende dieses Anhangs einige Simulationsbeispiele mit realen Außentemperaturverläufen, die einen Einblick in die Leistungsfähigkeit der Simulation geben.

A.1 Technische Beschreibung

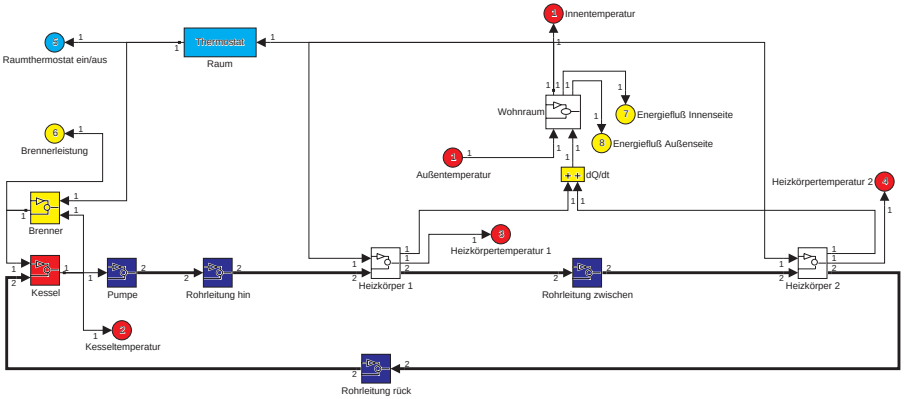


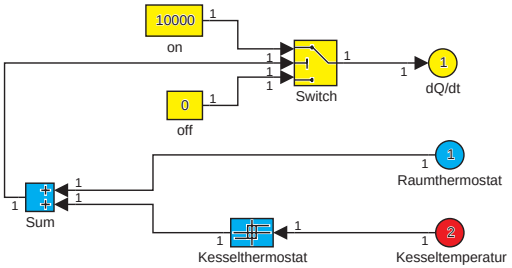
Abbildung A.1: Überblick über die Komponenten der Gebäudeheizungssimulation

Die komplette Simulation einer Heizung wie in Abbildung A.1 gezeigt, setzt sich zusammen aus einem Brenner mit zugehörigem Heizkessel, der Pumpe mit Rohrleitung, mehreren Heizkörpern bestehend aus Heizungsthermostat und Mischer und dem Wohnraum mit Fenstern, Türen und der Wand.

In den Grafiken wurden folgende farbliche Kennzeichnungen verwendet:

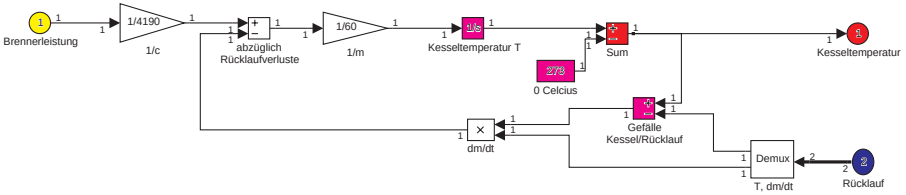
Rot	Temperatur in Grad Celsius
Magenta	Temperatur bzw. Temperaturdifferenz in Kelvin
Gelb	Energiefluß bzw. Leistung in Watt
Blau	Wassermassentransport $\frac{dm}{dt}$ in $\frac{kg}{s}$, Temperatur T in Grad Celsius
Cyan	Schaltausgänge
Schwarz	Keine spezielle Bedeutung

A.1.1 Brenner



Der Brenner liefert eine konstante Leistung $\dot{Q}_{Brenner}$ (15000 Watt). Die Regelung erfolgt einerseits über die Vorgabe einer Kesseltemperatur (55° Celsius), andererseits über einen Raumthermostaten.

A.1.2 Heizkessel



Im Kessel wird durch die Brennerleistung $\dot{Q}_{Brenner}$ das Heizwasser der Masse $\dot{m}$ erhitzt. Dabei wird berücksichtigt, daß sich das aus dem Heizungssystem zurücklaufende Wasser mit dem im Kessel befindlichen mischt. Die Formeln lauten:

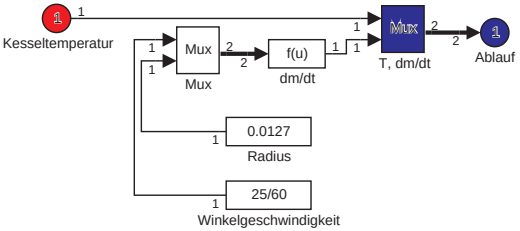
$$\begin{aligned} \dot{Q}_{Kessel} &= \dot{Q}_{Brenner} + \dot{Q}_{Rücklauf} - \dot{Q}_{Ablauf} \\ &= \dot{Q}_{Brenner} - \dot{m} \cdot c_{Wasser} \cdot (T_{Ablauf} - T_{Rücklauf}) \end{aligned} \tag{A.1}$$

$$\begin{aligned} T_{Kessel} &= \int \frac{\dot{Q}_{Kessel}}{m \cdot c_{Wasser}}\, dt \\ &= \int \frac{\dot{Q}_{Brenner}}{m \cdot c_{Wasser}} - \frac{\dot{m} \cdot (T_{Ablauf} - T_{Rücklauf})}{m}\, dt \end{aligned} \tag{A.2}$$

Formeln (A.1) und (A.2) verwenden folgende Bezeichnungen:

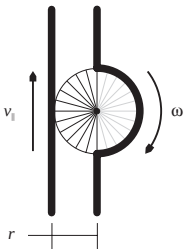
c_{Wasser}	Spezifische Wärmekapazität von Wasser	$\approx 4.19 \frac{kJ}{kgK}$
m	Zu erhitzende Wassermasse	
$\dot{m}$	Durch die Pumpe erzeugter Massenstrom	
T	Temperatur in Kelvin	
$\dot{Q}$	Leistung	

A.1.3 Pumpe



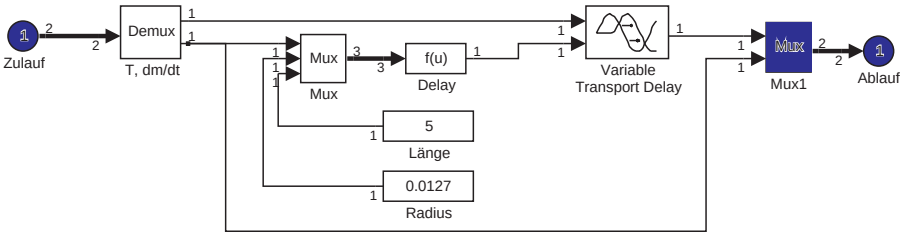
Die Pumpe befördert Wasser der Kesseltemperatur T mit dem Massenstrom $\dot{m}$. In einem Zeitraum Δt wird die Wassermasse $\dot{m}\Delta t$ befördert. Der Massenstrom errechnet sich aus dem Rohrdurchmesser r und der Pumpendrehzahl ω:

$$\begin{aligned} \dot{m} &\approx \bar{v}_{\parallel} \cdot r^2 \pi \cdot \rho \\ \bar{v}_{\parallel} &= \frac{r\pi}{2}\omega \end{aligned} \tag{A.3}$$



Die Bezeichnungen in Formel (A.3) sind:

ρ_{Wasser}	Dichte von Wasser	$\approx 1000 \frac{kg}{m^3}$
r	Rohrradius	$\approx \frac{1}{2}''$
$\dot{m}$	Massenstrom	
ω	Winkelgeschwindigkeit	
$\dot{Q}$	Kreiszahl Pi	≈ 3.1415
$\bar{v}_{\parallel}$	Mittlere Strömungsgeschwindigkeit	



A.1.4 Leitung

Die Modellierung der Verbindungsleitungen geht von ideal isolierten Rohren aus. Es ist nur die Zeitverzögerung

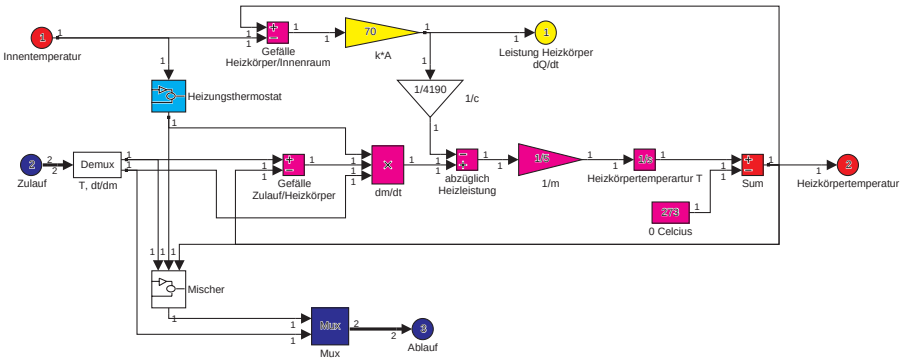
$$\tau = \frac{\rho_{Wasser} \cdot r^2 \pi \cdot l}{\dot{m}} \tag{A.4}$$

realisiert, die das Wasser vom Rohreintritt bis zum Rohraustritt benötigt, nicht aber der Temperaturverlust an die Umgebung.

Formel (A.4) verwendet folgende Bezeichnungen:

ρ_{Wasser}	Dichte von Wasser	$\approx 1000 \frac{kg}{m^3}$
r	Rohrradius	$\approx \frac{1}{2}''$
l	Rohrlänge	
$\dot{m}$	Massenstrom	
$\dot{Q}$	Kreiszahl Pi	≈ 3.1415

A.1.5 Heizkörper



Das Wasser im Heizkörper wird mit dem zulaufenden Heizwasser gemischt und dadurch erwärmt. Dabei wird berücksichtigt, daß durch einen Heizungsthermostaten unter Umständen nur ein Teil des Heizwassers in den Heizkörper gelangt, und der Rest direkt in die Zirkulationsleitung rückgespeist wird.

Aufgrund des Temperaturgefälles zwischen Heizkörper und Innenraum, kommt es zusätzlich noch zu einem Energiefluß zwischen diesen. Hierbei gelten folgende Beziehungen:

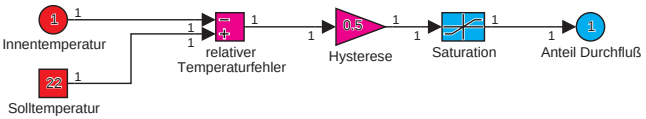
$$\begin{aligned} \dot{Q}_{Heizkörper} &= \dot{Q}_{Zulauf} - \dot{Q}_{Ablauf} - \dot{Q}_{Heizleistung} \\ &= \lambda \cdot \dot{m} \cdot c_{Wasser} \cdot (T_{Zulauf} - T_{Ablauf}) - \\ &\quad -k \cdot A_{Heizkörper} \cdot (T_{heizkörper} - T_{Wohnraum}) \end{aligned} \tag{A.5}$$

$$\begin{aligned} T_{Heizkörper} &= \int \frac{\dot{Q}_{Heizkörper}}{m \cdot c_{Wasser}}\, dt \\ &= \int \lambda \frac{\dot{m}(T_{Zulauf} - T_{Ablauf})}{m}\, dt - \\ &\quad - \int \frac{k \cdot A_{Heizkörper} \cdot (T_{Heizkörper} - T_{Wohnraum})}{m \cdot c_{Wasser}}\, dt \end{aligned} \tag{A.6}$$

Formeln (A.5) und (A.6) verwenden folgende Bezeichnungen:

c_{Wasser}	Spezifische Wärmekapazität von Wasser	$\approx 4.19 \frac{kJ}{kgK}$
m	Zu erhitzende Wassermasse	
$\dot{m}$	Durch die Pumpe erzeugter Massenstrom	
T	Temperatur in Kelvin	
$\dot{Q}$	Leistung	
A	Am Wärmeaustausch beteiligte Oberfläche	
k	Wärmedurchgangskoeffizient	
λ	Anteil des Wasserdurchflußes	$\in [0 \equiv \text{aus}, 1 \equiv \text{ein}]$

Heizkörperthermostat

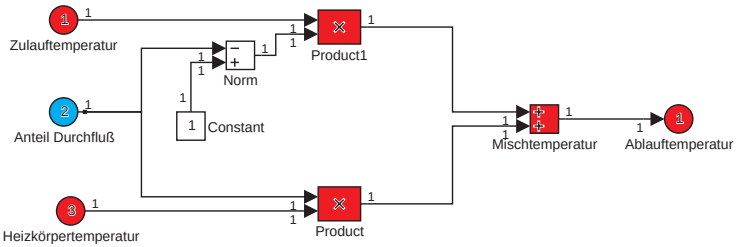


Der Thermostat regelt den Durchfluß des Heizwassers durch den Heizkörper im Hyseresebereich ϕ linear von 0% bis 100%. Ist die Innentemperatur größer oder gleich der Solltemperatur, so beträgt der Durchflußanteil 0%. Liegt die Innentemperatur mehr als ϕ Grad unter der Solltemperatur, so ist er 100%. Die korrespondierende Formel lautet:

$$\lambda = max(min(T_{Soll} - T_{Wohnraum}), 0), \phi) \tag{A.7}$$

Die Bezeichnungen in Formel (A.7) sind:

ϕ	Hysterese des Thermostaten	$\approx 2°$
T	Temperatur in Kelvin	
λ	Anteil des Wasserdurchflußes	$\in [0 \equiv \text{aus}, 1 \equiv \text{ein}]$



Mischer

Der Mischer berechnet die Temperatur des ablaufenden Wassers anhand des Durchflußanteils λ des Heizwassers. Beträgt der Anteil 0%, so fließt das Wasser komplett am Heizkörper vorbei und hat Heizwassertemperatur. Ist der Thermostat komplett geöffnet (Durchflußanteil 100%), so fließt nur noch Wasser des Heizkörpers ab. Die folgende Formel basiert auf dem zugrunde liegenden Energiefluß:

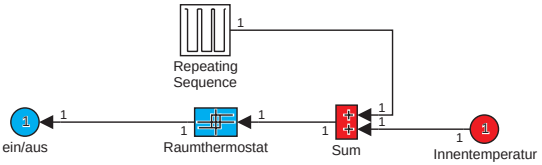
$$T_{Ablauf} = \lambda T_{Heizkörper} + (1 - \lambda) T_{Zulauf} \tag{A.8}$$

Die Bezeichnungen in Formel (A.8) sind:

T Temperatur in Kelvin

λ Anteil des Wasserdurchflußes $\in [0 \equiv \text{aus}, 1 \equiv \text{ein}]$

A.1.6 Raumthermostat

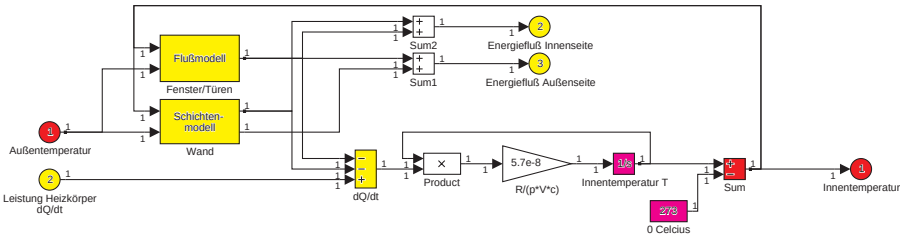


Der Raumthermostat schaltet den Brenner des Kessels bei Erreichen einer Solltemperatur ab. Ist die Innentemperatur größer oder gleich der Solltemperatur, so ist der Brenner aus. Liegt die Innentemperatur mehr als 1 Grad unter der Solltemperatur, so wird der Brenner wieder aktiviert. Der Solltemperaturverlauf kann explizit vorgegeben werden, wodurch eine Nachtabsenkung realisiert wird.

A.1.7 Wohnraum

Im Wohnraum wird durch die Heizkörperleistung $\dot{Q}_{Heizkörper}$ das Gasvolumen V erhitzt. Dabei wird die Ausdehnung des Gases während der Erwärmung und der Energiefluß durch Wand und Fenster bzw. Türen berücksichtigt. Die Formeln lauten:

$$\begin{aligned} \dot{Q}_{Wohnraum} &= \dot{Q}_{Heizkörper} - \dot{Q}_{Verlust} \\ &= \dot{Q}_{Heizkörper} - \dot{Q}_{Wand} - \dot{Q}_{Fenster/Türen} \end{aligned} \tag{A.9}$$

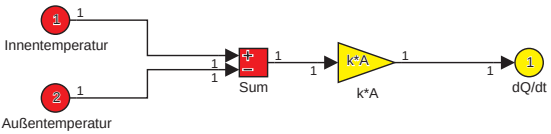


$$T_{Wohnraum} = \int \frac{R_{Luft} T_{Wohnraum}}{p \cdot V_{Wohnraum} \cdot c_{Luft}} \dot{Q}_{Wohnraum} dt \tag{A.10}$$

Formeln (A.9) und (A.10) verwenden folgende Bezeichnungen:

c_{Luft}	Spezifische Wärmekapazität von Luft (p=const)	$\approx 1.009 \frac{kJ}{kgK}$
R_{Luft}	Gaskonstante von Luft	$\approx 287 \frac{J}{kgK}$
$V_{Wohnraum}$	Beteiligtes Gasvolumen	
T	Temperatur in Kelvin	
$\dot{Q}$	Leistung	
A	Am Wärmeaustausch beteiligte Oberfläche	
k	Wärmedurchgangskoeffizient	
p	Luftdruck	

Fenster und Türen



Da Fenster und Türen eine zu vernachlässigende Wärmekapazität besitzen, wird in der Simulation ein stationärer Wärmeenergiefluß angenommen. Es ergibt sich somit:

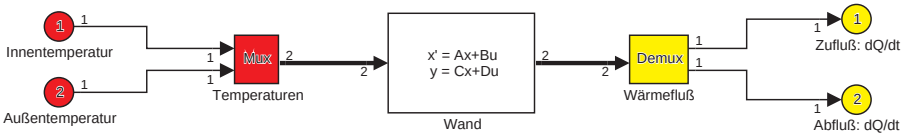
$$\dot{Q}_{Fenster/Türen} = k \cdot A_{Fenster/Türen} \cdot (T_{Wohnraum} - T_{Außen}) \tag{A.11}$$

Formel (A.11) verwendet folgende Bezeichnungen:

T	Temperatur in Kelvin
$\dot{Q}$	Leistung
A	Am Wärmeaustausch beteiligte Oberfläche
k	Wärmedurchgangskoeffizient

Wand

Zur Simulation der Wand wurde ein Schichtenmodell verwendet. Die Wand wird dabei in n Schichten unterteilt. Die Schicht i besitzt dabei die Dicke d_i, die momentane Temperatur T_i



und den Wärmewiderstand λ_i. Der Wärmeenergiefluß von der Schicht i zur Schicht $i+1$ wird mit $\dot{Q}_{i/i+1}$ bezeichnet und ist somit positiv, wenn $T_i > T_{i+1}$ gilt. Die folgende Abbildung verdeutlicht nochmals das zugrundegelegte physikalische System.

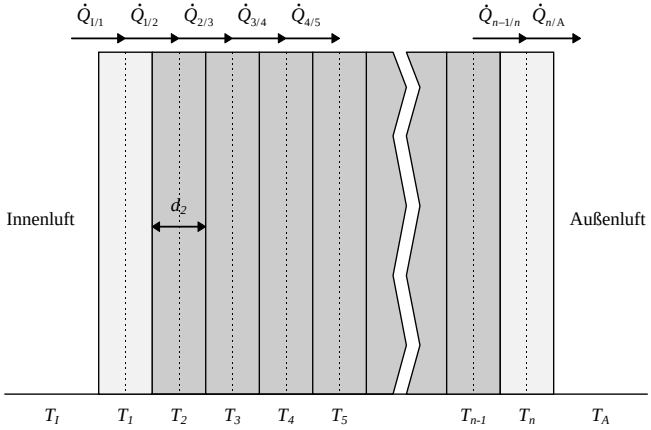


Abbildung A.2: Schichtenmodell einer Gebäudewand

Für den Wärmeenergiefluß $\dot{Q}_{i/i+1}$ und die Temperatur T_i gilt nun:

$$\dot{Q}_{i/i+1} = \underbrace{\frac{2\lambda_i\lambda_{i+1}}{\lambda_{i+1}d_i + \lambda_i d_{i+1}}}_{\beta_i} \cdot A_{Außenwand} \cdot (T_i - T_{i+1}) \tag{A.12}$$

$$T_i = \int \frac{\dot{Q}_{i-1/i} - \dot{Q}_{i/i+1}}{A_{Außenwand} \cdot d_i \rho_i c_i}\, dt, \quad i \in \{1, n_{Schichten}\} \tag{A.13}$$

Aus Formel (A.12) und (A.13) läßt sich nun folgendes Differentialgleichungssystem herleiten:

$$\dot{T}_i = \frac{1}{d_i \rho_i c_i}(\beta_{i-1}T_{i-1} - (\beta_{i-1} + \beta_i)T_i + \beta_i T_{i+1}), \quad i \in \{1, n_{Schichten}\} \tag{A.14}$$

Will man nun das System (A.14) in der Form $\dot{T} = \mathbf{A} \cdot T + \mathbf{B} \cdot U$ darstellen, so erhält man für die Matrizen $\mathbf{A}$ und $\mathbf{B}$:

$$\mathbf{A} \equiv \begin{pmatrix} -(\beta_0+\beta_1) & \beta_1 & & & \\ \beta_1 & -(\beta_1+\beta_2) & \beta_2 & & \\ & \ddots & & & \\ & \beta_{n-2} & -(\beta_{n-2}+\beta_{n-1}) & \beta_{n-1} & \\ & & \beta_{n-1} & -(\beta_{n-1}+\beta_n) & \end{pmatrix}, \mathbf{B} \equiv \begin{pmatrix} \beta_0 & 0 \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ 0 & \beta_n \end{pmatrix} \tag{A.15}$$

Als Ausgabegrößen bieten sich nun der Wärmeenergiefluß $\dot{Q}_{I/1}$ in die Wand und $\dot{Q}_{n/A}$ aus der Wand an:

$$\begin{aligned} \dot{Q} &= \mathbf{C} \cdot T + \mathbf{D} \cdot U \\ \mathbf{C} &= \begin{pmatrix} -\beta_0 \cdot A_{Innenwand} & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & 0 & \beta_n \cdot A_{Außenwand} \end{pmatrix} \\ \mathbf{D} &= \begin{pmatrix} \beta_0 \cdot A_{Innenwand} & 0 \\ 0 & -\beta_n \cdot A_{Außenwand} \end{pmatrix} \end{aligned} \tag{A.16}$$

Die Formeln (A.12) bis (A.16) verwenden folgende Bezeichnungen:

d	Schichtdicke
λ	Wärmewiderstand
ρ	Dichte
T	Temperatur in Kelvin
$\dot{Q}$	Wärmeenergiefluß (Leistung)
A	Am Wärmeaustausch beteiligte Oberfläche
c	Spezifische Wärmekapazität

A.2 Simulationsbeispiele

A.2.1 Energiefluß durch eine Wand bei konstanter Innen- und Außentemperatur

Für diese Simulation wurde eine konstante Ausgangswandtemperatur von 20°C angenommen. Anschließend fällt die Außentemperatur sprunghaft auf 10°C. Abbildung A.3 zeigt den Wärmeenergiefluß in die Wand hinein, bzw. aus der Wand heraus. Die Innentemperatur wird konstant angenommen, was einem beliebig großem Wärmereservoir entspricht. Die Wand wurde mit 50 Schichten simuliert. Wie ersichtlich ist, tritt ein nennenswerter Energiefluß aufgrund des Außentemperatursprungs erst nach etwa einem halben Tag auf. Der stationäre Zustand wird nach ca. 4 Tagen erreicht, wie auch aus dem zeitlichen Verlauf des Wandtemperaturprofils ersichtlich ist.

In Abbildung A.4 ist der zeitliche Verlauf des Wandtemperaturprofils zu sehen. Man erkennt den stationären Zustand nach ca. 4 Tagen, der einem konstanten Wärmefluß entspricht. Auch hier zeigt sich die Verzögerung von etwa $\frac{1}{2}$ Tag, bis Außentemperaturschwankungen zum Innenraum vordringen.

Zu bemerken ist, daß im realen Gebäude natürlich noch der Energiefluß durch Fenster und Türen zu berücksichtigt werden muß, welche eine zu vernachlässigende Wärmekapazität besitzen.

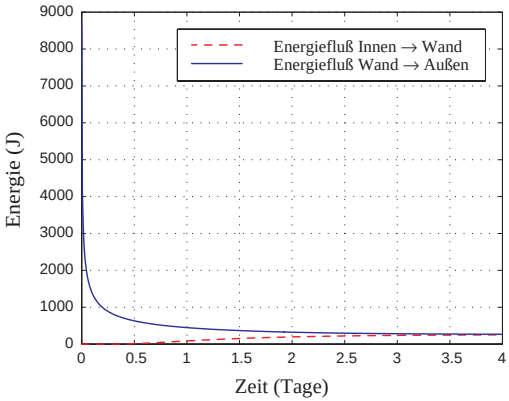


Abbildung A.3: Energiefluß durch eine Gebäudeaußenwand (Temperatursprung)

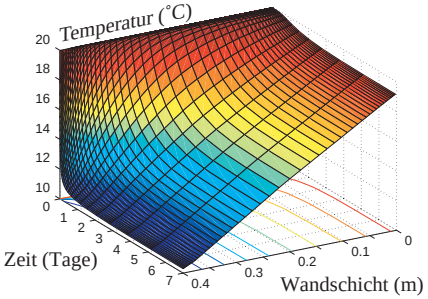


Abbildung A.4: Temperaturverteilung in einer Gebäudeaußenwand (Temperatursprung)

A.2.2 Ein kompletter Zwei-Tages Heiz-Zyklus

Die Heizungsregelung besteht aus einem Raumthermostaten, der bei Temperaturen unter 20°C den Brenner einschaltet und mit einer Hysterese von 1°C behaftet ist. Die Heizkörperthermostate regeln davon unabhängig und schließen den Warmwasserzufluß bei 23°C. Der Kesselthermostat ist so eingestellt, daß er die Kesseltemperatur über 50°C hält. Seine Hysterese liegt bei ca. 5°C. Der Temperaturverlauf der Außentemperatur orientiert sich am Profil eines normalen Herbsttages. Die eingesetzten Konstanten charakterisierten die übliche Zentralheizung eines Einfamilienhauses mit einem Gasbrenner.

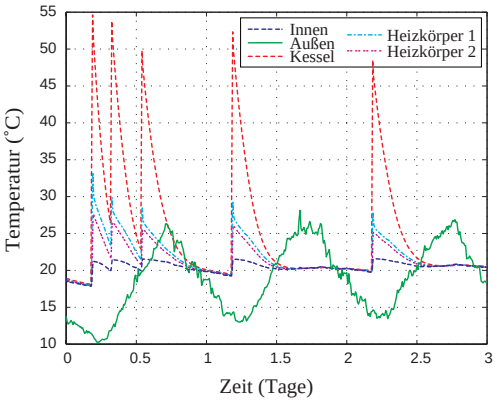


Abbildung A.5: Temperatur verschiedener Heizungskomponenten (Zwei-Tages Zyklus)

Abbildung A.5 zeigt den Temperaturunterschied zwischen Kesselwasser und Wassertemperatur im ersten Heizkörper. Dies ist auf den Mischvorgang mit dem noch kalten Wasser in Heizkreislauf zurückzuführen. Der Temperaturunterschied zwischen beiden Heizkörpern ist mit dem Aufheizen der Raumluft verbunden. Zwischen den einzelnen Komponenten befinden sich verzögernde Rohrleitungen, was aus den Temperaturverläufen abgelesen werden kann.

Der Energiefluß während der Simulationphase vom Wohnraum in die Wand bzw. von der Wand in die Umgebung ist in Abbildung A.6 gezeigt. Da die Wand über keine eigenständige Energiequelle verfügt, muß aufgrund des Energieerhaltungssatzes ein Gleichgewicht gelten, was sich auch an der Grafik ablesen läßt. Die Wärmekapazität der Wand puffert den Einfluß der Außentemperaturschwankungen stark ab und verzögert diese. Für eine Heizstrategie bedeutet dies, daß sich diese nicht nach der aktuellen Außentemperatur richten, sondern explizit den Energiefluß in die Wand und durch die Türen und Fenster modellieren sollte.

In Abbildung A.7 ist der zeitliche Verlauf des Wandtemperaturprofils zu sehen. Man erkennt auch hier, daß die Wärmekapazität der Wand den Einfluß der Außentemperaturschwankungen abpuffert und um etwa $\frac{1}{2}$ Tag verzögert. Im Mauerkern treten nur minimale Temperaturschwankungen um ca. 8°C herum auf. Auch hier läßt sich ablesen, daß eine Außentemperaturnachführung nicht sinnvoll ist. Um eine optimale Regelung zu erreichen darf die Wärmekapazität der Wände nicht vernachlässigt werden. Eine optimale Strategie muß exakt

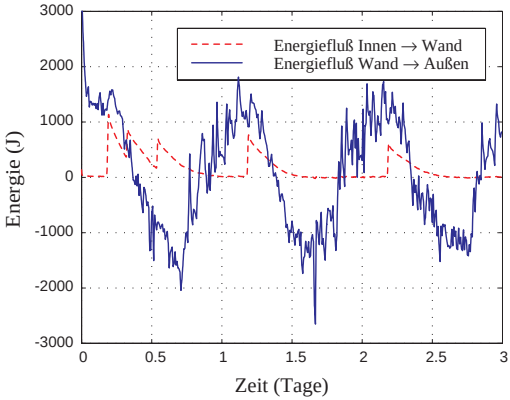


Abbildung A.6: Energiefluß durch eine Gebäudeaußenwand (Zwei-Tages Zyklus)

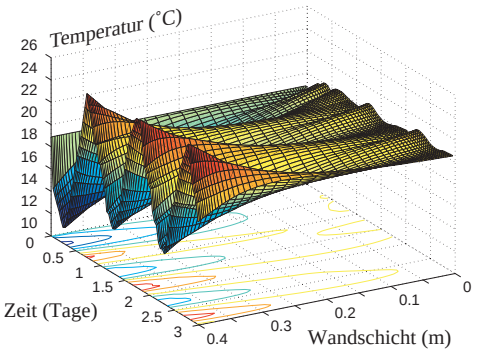


Abbildung A.7: Temperaturverteilung in einer Gebäudeaußenwand (Zwei-Tages Zyklus)

die Wärmemenge produzieren, die nötig ist, um den Energieverlust durch Wände, Türen und Fenster zu kompensieren.

Anhang B

LEMON-Algorithmus

Der folgende Algorithmus ist als Teil der AMoC Biliothek [100] realisiert worden.

repeat N **times**
  if Ausnahmen-Liste ist voll
    Generiere n_{ce} neue Codebuchvektoren $\vec{c}$
    Selektiere zugehörige Prototypmodelle $\mathfrak{M}$ anhand gegebener Regelbasis
    Füge neue Ellipsoide $\mathbf{E}$ in Codebuch ein
  end if
  for Alle Datenvektoren $\vec{x}$
    Finde k mit $\forall i \neq k : \mathfrak{D}_i(\vec{x}) \geq \mathfrak{D}_k(\vec{x})$
    Berechne Fehler der Modellvorhersage $\Delta_{\vec{y}} = \|\mathfrak{M}_k(\vec{x}) - \vec{y}\|$
    if $\vec{x}$ ist *unbekannt*
      if $\Delta_{\vec{y}} \leq \epsilon$
        Ersetze $\vec{c}_k$ durch $\vec{c}_k + \beta(\vec{x} - \vec{c}_k)$
        Ersetze $\mathbf{E}_k$ durch $\mathbf{E}_i + (\alpha - 1)\vec{a}\vec{a}^T\mathbf{E}_i$
        Optional: Modifiziere α_k nach gewünschter Methode
        Aktualisiere Konfidenz K_k anhand von $\Delta_{\vec{y}}$
      else
        Speichere $(\vec{x}, \vec{y})$ in Ausnahmen-Liste
      end if
    else
      Aktualisiere Konfidenz K_k anhand von $\Delta_{\vec{y}}$
      if $\Delta_{\vec{y}} \leq \epsilon \vee K_{bad} \leq K_k$
        Trainiere $\mathfrak{M}_k$ mit $(\vec{x}, \vec{y})$
      elseif $K_{del} \leq K_k < K_{bad}$
        Speichere $(\vec{x}, \vec{y})$ in Ausnahmen-Liste
      elseif $K_k < K_{del}$
        Speichere $(\vec{x}, \vec{y})$ in Ausnahmen-Liste
        Lösche Codebucheintrag k
      end if
    end if
  end for
end repeat

Anhang C

Parameterbeschreibung

C.1 Simulationssteuerung von LEMON

Die folgenden Parameter können zur Steuerung des Verhaltens von LEMON modifiziert werden. Durch Setzen der korrespondierenden Umgebungsvariablen werden die Standardwerte überschrieben. Jeweils in eckigen Klammern finden sich die Standardwerte. Falls vorhanden finden sich Verweise auf diejenigen Textstellen, in denen die Parameter besprochen werden.

`LMODEL_TRAINLENGTH[100]:` Zahl der zu puffernden Ein-/Ausgabepaare zum Belernen eines lokalen Modells. Das loakle Modell wird erst bei Erreichen der hier spezifizierten Zahl von Ein-/Ausgabepaaren trainiert. Diese werden vor dem Training gemischt, und nach dem Training aus der Liste gelöscht.

`LMN_MAXEXCEPTIONS[100]:` Legt die maximale Länge der Ausnahmen-Liste fest, wie es auf Seite 67 erläutert ist. Bei Erreichen dieser Länge werden maximal `LMN_CEVECTORS` neue Ellipsoide generiert und eingefügt.

`LMN_MAXINITLENGTH[100]:` Maximale Anzahl von Ein-/Ausgabepaaren die zur Modellinitialisierung genutzt werden. Da bei der Initialisierung einiger Modelle eine Pseudoinverse zu berechnen ist, sollte diese Länge nicht zu groß gewählt werden.

`LMN_CEVECTORS[4]:` Zahl der maximal einzufügenden neuen Ellipsoide.

`LMN_EDIST[0.3]:` Initialer Radius für neu eingefügte Ellipsoide und maximal zulässiger Abstand von der Ellipsoidoberfläche, falls eine räumliche Adaption erfolgen soll. Entspricht $d_{\min}$ auf Seite 66.

`LMN_MINRADIUS[`$0.2 \cdot$ `LMN_EDIST]:` Falls ein Ellipsoid innerhalb eines anderen Ellipsoids eingefügt wird, wird sein initialer Radius so angepaßt, daß es komplett innerhalb liegt und keine Überschneidungen mit anderen Ellipsoiden vorhanden sind. Ellipsoide deren Radius dadurch unter die hier angegebene Schranke fallen, werden nicht erzeugt.

`LMN_APLHA[0.3]:` Empfohlene Schrittweite für Kohonens OLVQ1 Algorithmus. Wird zur Initialisierung von α_k auf Seite 67 verwendet.

`LMN_STEPS[`$50 \cdot$ `LMN_CEVECTORS]:` Zahl der Durchläufe für Kohonens OLVQ1 Algorithmus. Entspricht N auf Seite 67.

`LMN_BETA[0.1]:` Verschiebungsfaktor zur räumlichen Adaption der Ellipsoide. Es ist die Variante mit festem α implementiert, wie sie auf Seite 68 beschrieben wird.

`LMN_MAXERROR[0.025]:` Gibt den kleinsten zu berücksichtigenden Vorhersage-Fehler an. Kleinere Fehler werden mit 0 gleichgesetzt. Entspricht ϵ auf Seite 66.

`LMN_BADCONF[0.5]:` Konfidenzschranke ab der die Generierung von Subellipsoiden zugelassen wird. Entspricht K_{bad} auf Seite 66.

`LMN_DELCONF[0.25]:` Konfidenzschranke ab der ein Ellipsoid und das zugehörige Modell gelöscht werden. Entspricht K_{del} auf Seite 66.

C.2 Fuzzy-Variablen

Im folgenden findet sich ein Aufstellung der zur Zeit im Fuzzy-Modul von LEMON nutzbaren Variablen.

C.2.1 Modellspezifische Variablen

`nParams:` Gibt die Zahl der zu optimierenden Modell-Parameter wieder und ist somit ein Maß für die Komplexität des Modells.

`avgDist:` Enthält die Summe der Fehlerquadrate bei Vorhersage der zur Ellipsoidgenerierung genutzten Ein-/Ausgabepaare der Ausnahmen-Liste. Die Berechnung erfolgt mittels Crossvalidation (siehe Seite 19) und erlaubt somit eine erste Aussage über die Eignung des Modells für den präsentierten Datensatz.

C.2.2 Datenspezifische Variablen

`{min,max,avg}VarTarget:` Die minimale, maximale und mittlere Varianz über alle zur Ellipsoidgenerierung genutzten Ausgabedaten der Ausnahmen-Liste. Die Minimierung, Maximierung und Mittelung erfolgt dabei über die unterschiedlichen Ausgabedimensionen.

`{min,max,avg}VarInput:` Die minimale, maximale und mittlere Varianz über alle zur Ellipsoidgenerierung genutzten Eingabedaten der Ausnahmen-Liste. Die Minimierung, Maximierung und Mittelung erfolgt dabei über die unterschiedlichen Eingabedimensionen.

Literatur

[1] Åström, K. J. und Wittenmark, B.: *Adaptive Control*. Control Engineering: Addison Wesley 1989.

[2] Ayoubi, M.: Das dynamische Perzeptronmodell zur experimentellen Modellbildung nichtlinearer Prozesse. *Informatik Forschung und Entwicklung*, S. 14–22.

[3] Ayoubi, M. und Isermann, R.: Radial basis function networks with distributed dynamics for nonlinear dynamic system identification. In *Third European Congress on Intelligent Techniques and Soft Computing, EUFIT'95*, 1995.

[4] Baker, W. L. und Farrell, J. A.: An introduction to connectionist learning control systems. In White, D. A. und Sofge, D. A. (Hrsg.), *Handbook of Intelligent Control*. VAN NOSTRAND REINHOLD, New York, 1992, Kap. I, S. 35–63.

[5] Bates, D. M. und Watts, D. G.: *Nonlinear Regression Analysis and its Applications*. New York: Wiley 1988.

[6] Bronstein, I., Semendjajew, K., Musiol, G., und Muehlig, H.: *Taschenbuch der Mathematik*. Thun: Harri Deutsch Verlag 1993.

[7] Brychcy, T.: Trajektoriengenerierung durch Trajektorienspeicherung mittels adaptiver, neuronaler Raumrepräsentation. Diplomarbeit, Prof. Dr. Brauer, Institut für Informatik, Lehrstuhl Prof. Dr. Dr. h.c. W. Brauer, Technische Universität München 1994.

[8] Brychcy, T.: Vorstrukturierte Verallgemeinerte Rekurrente Neuronale Netze. *Forschungsberichte Künstliche Intelligenz: FKI-223-97*, Hrsg. Institut für Informatik, Technische Universität München 1997.

[9] Brychcy, T.: Prestructured recurrent neural networks. In Brauer, W. (Hrsg.), *Fuzzy-Neuro Systems ´98 - Computational Intelligence*, Proceedings in Artificial Intelligence. Infix Verlag, St. Augustin, 1998, S. 210–217.

[10] Brychcy, T.: *Modellierung dynamischer Systeme mit vorstrukturierten neuronalen Netzen*. Berlin: Akad. Verl.-Ges. Aka 2000. (Dissertationen zur künstlichen Intelligenz; Bd. 229.) Zugl.: München, Techn. Univ., Diss., 1999. ISBN 3-89838-229-X.

[11] Brychcy, T. und Kinder, M.: A neural network inspired architecture for robot motion planning. In Bulsari, A. und Kallio, S. (Hrsg.), *Engineering Applications of Artificial Neural Networks: Proceedings of the International Conference EANN '95, 21-23 August 1995, Otaniemi/Helsinki, Finland*. Finnish Artificial Intelligence Society, 1995, S. 103–110.

[12] Buchberger, B.: Gröbner bases: An algorithmic method in polynomial ideal theory. In Bose, N. K. (Hrsg.), *Multidimensional Systems Theory*. K. Reidel Publishing Company, Dordrecht, 1985, S. 184–232.

[13] Buchberger, B.: A note on the complexity of constructing Gröbner- bases. In van Hulzen, J. A. (Hrsg.), *Proc. European Computer Algebra Conference, EUROCAL '83, LNCS 162*. Springer Verlag, London, 1983, S. 137–145.

[14] Butz, D.: *Neuronale Funktionsapproximation mit RBF-Schwerpunktnetzen*. Aachen: Shaker Verlag 1997.

[15] Camacho, E. F. und Bordons, C.: *Model Predictive Control in the Process Industry*. London: Springer Verlag 1999.

[16] Cohn, D. A.: Neural network exploration using optimal experiment design. In Cowan, J. D., Tesauro, G., und Alspector, J. (Hrsg.), *Advances in Neural Information Processing Systems*, Bd. 6. Morgan Kaufmann Publishers, Inc., 1994, S. 679–686.

[17] Cun, Y. L., Denker, J. S., und Solla, S. A.: Optimal brain damage. In Touretzky, D. S. (Hrsg.), *Advances in Neural Information Processing Systems*, Bd. 2. Morgan Kaufmann, San Mateo, CA, 1990, S. 598–605.

[18] Dasarathy, B. V. (Hrsg.): *Nearest Neighbor Pattern Classification Techniques*. IEEE Computer Society Press 1991.

[19] Day, S. P. und Davenport, M. R.: Continous-time temporal back-propagation with adaptable time delays. *IEEE Transactions on Neural Networks*, *Bd. 4 [1993]* Nr. 2, S. 348–354. preprint available since 1991 via `ftp://archive.cis.ohio-state.edu/pub/neuroprose/day.temporal.ps`.

[20] Deco, G. und Obradovic, D.: *An Information-Theoretic Approach to Neural Computing*. New-York Berlin Heidelberg: Springer 1996.

[21] Eder, K.: *Repräsentation temporaler Information in neuronalen Netzen*. Dissertation, Institut für Informatik, Lehrstuhl Prof. Dr. Dr. h.c. W. Brauer, Technische Universität München 1995. VDI Verlag, Düsseldorf.

[22] Efron, B.: *The Jackknife, the Bootstrap and Other Resampling Plans*, Bd. 38. Society for Industrial and Applied Mathematics (SIAM); SERIES:CBMS-NSF Regional Conference Series in Appli 1982.

[23] Efron, B.: Estimating the error rate of a prediction rule: improvement on cross-validation. *Journal of the American Statistical Association*, *Bd. 78 [1983]* Nr. 382, S. 316–331.

[24] Efron, B.: More efficient bootstrap computations. *Journal of the American Statistical Association*, *Bd. 85 [1990]* Nr. 409, S. 79–89.

[25] Elman, J. L.: Finding structure in time. *Technischer Bericht: CRL-8801*, Hrsg. Center for Research in Language, UCSD 1988.

[26] Fahlman, S.: An empirical study of learning speed in back-propogation networks. *Technical Report*, Hrsg. Neuroprose. availabel via `ftp://archive.cis.ohio-state.edu/pub/neuroprose/fahlman.quickprop-tr.ps.Z` 1988.

[27] Fedorov, V. V.: *Theory of Optimal Experiments*. New York: Academic Press 1972.

[28] Ferber, J.: *Multi-Agent Systems. An Introduction to Distributed Artificial Intelligence*. New York: John Wiley & Sons Inc. 1999.

[29] Föllinger, O.: *Regelungstechnik*. 7. Auflage. Heidelberg: Hüthig 1992.

[30] Franklin, S. und Graesser, A.: Is it an agent, or just a program?: A taxonomy for autonomous agents. In Müller, J., Wooldridge, M., und Jennings, N. (Hrsg.), *Intelligent Agents III*, Lecture Notes in Artificial in Artificial Intelligence, Vol. 1193. Springer-Verlag, Berlin et al., 1997, S. 21–36.

[31] French, R. M.: Catastrophic forgetting in connectionist networks: Causes, consequences and solutions. *Trends in Cognitive Sciences*. to appear.

[32] Garcia, C. E., Prett, D. M., und Morari, M.: Model predictive control: Theory and practice – A survey. *Automatica*, *Bd. 25 [1989]* Nr. 3, S. 335–348.

[33] Geman, S., Bienenstock, E., und Doursat, R.: Neural networks and the bias/variance dilemma. *Neural Computation*, *Bd. 4 [1992]* Nr. 1, S. 1–58.

[34] Golub, G. H. und Loan, C. F. V.: *Matrix Computations*. 2. Auflage., Bd. 3 erschienen in*Johns Hopkins Series in the Mathematical Sciences*. Baltimore, MD, USA: The Johns Hopkins University Press 1989.

[35] Goonatilake, S. und Khebbal, S.: Intelligent hybrid systems: Issues, classifications and future directions. In Goonatilake, S. und Khebbal, S. (Hrsg.), *Intelligent Hybrid Systems*. John Wiley & Sons Ltd., Chichester, 1995, S. 1–20.

[36] Hanson, S. J. und Pratt, L.: A comparison of different biases for minimal network construction with back-propagation. In Touretzky, D. S. (Hrsg.), *Advances in Neural Information Processing Systems*, Bd. 1. Morgan Kaufmann, San Mateo, CA, 1989, S. 177–185.

[37] Hardin, R. H. und Sloane, N. J. A.: A new approach to the construction of optimal designs. *Journal of Statistical Planning and Inference*, *Bd. 37* [1993], S. 339–369.

[38] Hardin, R. H. und Sloane, N. J. A. 1994. *OPERATING MANUAL FOR GOSSET: A GENERAL-PURPOSE PROGRAM FOR CONSTRUCTING EXPERIMENTAL DESIGNS*. 2. Auflage. Mathematical Sciencies Research Center, AT&T Bell Laboratories, Murray Hill, New Jersey.

[39] Hassibi, B. und Stork, D. G.: Second derivatives for network pruning: Optimal Brain Surgeon. In Hanson, S. J., Cowan, J. D., und Giles, C. L. (Hrsg.), *Advances in Neural Information Processing Systems*, Bd. 5. Morgan Kaufmann, San Mateo, CA, 1993, S. 164–171.

[40] Hochreiter, S.: *Generalisierung bei Neuronalen Netzen geringer Komplexität*. Dissertation, Institut für Informatik, Lehrstuhl Prof. Dr. Dr. h.c. W. Brauer, Technische Universität München 1999. Shaker Verlag, Aachen.

[41] Hornik, K., Stinchcombe, M., und White, H.: Multilayer feedforward networks are universal approximators. *Neural Networks*, *Bd. 2* [1989], S. 359–366.

[42] Huhns, M. und Singh, M. (Hrsg.): *Readings in Agents*. San Francisco, CA: Morgan Kaufmann 1998.

[43] Hunt, K. J., Sbarbaro, D., Żbikowski, R., und Gawthrop, P. J.: Neural networks for control systems – A survey. *Automatica, Bd. 28 [1992]* Nr. 6, S. 1083–1112.

[44] Jackson, L. B.: *Digital Filters and Signal Processing, with MATLAB Exercises*. 3. Auflage. Dordrecht, The Netherlands: Kluwer Academic Publishers Group 1996.

[45] Jacobs, R. A., Jordan, M. I., Nowlan, S. J., und Hinton, G. E.: Adaptive mixtures of local experts. *Neural Computation, Bd. 3 [1991]* Nr. 1, S. 79–87.

[46] Jordan, M. I.: Attractor dynamics and parallelism in a connectionist sequential machine. In *Proceedings of the Eighth Annual Cognitive Science Society Conference*. Erlbaum, Hillsdale, NJ, 1986.

[47] Jordan, M. I. und Jacobs, R. A.: Hierarchical mixtures of experts and the EM algorithm. *Neural Computation, Bd. 6 [1994]* Nr. 2, S. 181–214.

[48] Jörgl, H. P.: *Repetitorium Regelungstechnik*, Bd. 1. Wien, München: R. Oldenburg Verlag 1993.

[49] Kinder, M.: *Pfadplanung für Manipulatoren in komplexen Umgebungen mittels generalisierender Pfadspeicherung in Ellipsoidkarten*, Bd. 580 erschienen in*Fortschrittsberichte VDI, Reihe 8*. Düsseldorf: VDI Verlag 1996.

[50] Kirchmair, C.: Ein Gradientenabstiegsverfahren zum Schätzen der Parameter des Preisach Modells für Hysterese. *Forschungsberichte Künstliche Intelligenz: FKI-231-99*, Hrsg. Institut für Informatik, Technische Universität München 1999.

[51] Kohonen, T., Hynninen, J., Kangas, J., Laaksonen, J., und Torkkola, K. 1995. *LVQ_PAK, The Learning Vector Quantization Program Package, Version 3.1*. Helsinki University of Technology, Laboratory of Computer and Information Science, Rakentajanaukio 2 C, SF-02150 Espoo, FINLAND.

[52] Kohonen, T.: New developments of Learning vector Quantization and the Self-Organizing map. In *Symp. on Neural Networks; Alliances and Perspectives in Senri*. Senri Int. Information Institute, Osaka, Japan, 1992.

[53] Kohonen, T.: *Self-Organizing Maps*. 2. Auflage., Bd. 30 erschienen in*Springer Series in Information Sciences*. Berlin, Heidelberg: Springer 1997.

[54] Kriebel, S. K. T.: *A Combined Parametric and Nonparametric Approach to Time Series Analysis*. Dissertation, Institut für Informatik, Lehrstuhl Prof. Dr. Dr. h.c. W. Brauer, Technische Universität München 1999. Infix Verlag, St. Augustin.

[55] Kruse, R., Gebhardt, J., und Klawonn, F.: *Foundations of Fuzzy Systems*. Chichester: Wiley 1994.

[56] Lang, K. J. und Hinton, G. E.: A time-delay neural network architure for speech recognition. *Technical Report: CMU-CS-88-152*, Hrsg. Carnegie-Mellon University 1988.

[57] Leonhard, W.: *Einfuehrung in die Regelungstechnik*. Braunschweig: Vieweg 1992.

[58] Lin, D.-T., Dayhoff, J. E., und Ligomedes, P. A.: A learning algorithm for adaptive time-delays in a temporal neural network. *Technischer Bericht*, Hrsg. Univerity of Maryland 1992.

[59] Lin, D.-T., Dayhoff, J. E., und Ligomedes, P. A.: Trajectory production with the adaptive time-delay neural network. *Neural Networks*, *Bd. 8 [1993]* Nr. 3, S. 447–461.

[60] Mayergoyz, I.: *Mathematical Models of Hysteresis*. Berlin, Heidelberg, New York: Springer Verlag 1991.

[61] Miller, W. T., Sutton, R. S., und Werbos, P. J. (Hrsg.): *Neural Networks for Control*. Cambridge, Mass.: MIT Press 1990.

[62] Müller, K.-R., Kohlmorgen, J., und Pawelzik, K.: Analysis of switching dynamics with competing neural networks. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, *Bd. E78-A [1995]* Nr. 10, S. 1306–1315.

[63] Moody, J. E.: The effective number of parameters: An analysis of generalization and regularization in nonlinear learning systems. In Moody, J. E., Hanson, S. J., und Lippmann, R. P. (Hrsg.), *Advances in Neural Information Processing Systems*, Bd. 4. Morgan Kaufmann, San Mateo, CA, 1992, S. 847–854.

[64] Moody, J. und Darken, C. J.: Fast learning in networks of locally-tuned processing units. *Neural Computation*, *Bd. 1* [1989], S. 281–294.

[65] Moré, J. J.: The Levenberg-Marquardt algorithm: Implementation and theory. In *G.A. Watson*, Lecture Notes in Mathematics 630. Springer Verlag, Berlin, 1978, S. 105–116. Cited in Åke Björck's bibliography on least squares, which is available via `ftp://math.liu.se/pub/references`.

[66] Moré, J., Garbow, B., und Hillstrom, K.: User guide for minpack-1. *Technischer Bericht: Report ANL-80-74*, Hrsg. Argonne National Laboratory 1980.

[67] Murray-Smith, R. und Johansen, T. A.: Local learning in local model networks. In Murray-Smith, R. und Johansen, T. A. (Hrsg.), *Multiple model approaches to modelling and control*. Taylor & Francis, London, 1997.

[68] Narendra, K. S. und Parthasarathy, K.: Identification and control of dynamical systems using neural networks. *Proceedings of the IEEE First Annual International Conference on Neural Networks*, *Bd. 1 [1990]* Nr. 1, S. 4–27.

[69] Narendra, K. S. und Annaswamy, A. M.: *Stable Adaptive Systems*. Prentice Hall information and system sciences: Prentice-Hall 1989.

[70] Nauck, D., Klawonn, F., und Kruse, R.: *Foundations of Neuro–Fuzzy Systems*. Chichester: Wiley 1997.

[71] Nordhoff, S., Landorff, B., Labinsky, C., und Hartwig, R. 1995. *FOOL & FOX, Fuzzy System Development Tools, User Manual*. Universität Oldenburg, Oldenburg.

[72] O'Hare, G. und Jennings, N. (Hrsg.): *Foundations of Distributed Artificial Intelligence*. New York: John Wiley & Sons Inc. 1996.

[73] Opper, M.: A bayesian approach to on-line learning. In Saad, D. (Hrsg.), *On-line Learning in Neural Networks*, Publications of the Newton Institute. Cambridge University press, Cambridge, 1999, S. 363–378.

[74] Parks, T. W. und Burrus, C. S.: *Digital Filter Design.* 1. Auflage. New York, NY: John Wiley & Sons. Inc. 1987.

[75] Poggio, T. und Giorosi, F.: A theory of networks for approximation and learning. *A.I. Memo: No. 1140*, Hrsg. Massachusetts Institute of Technology, Artificial Intelligence Laboratory. available via `ftp://publications.ai.mit.edu/ai-publications/1000-1499/ATM-1140.ps.gz` 1989.

[76] Powell, M. J. D.: Radial basis functions for multivariable interpolation: A review. In *Proc. IMA Conf. on "Algorithms for the approximation of functions and data"*. RMCS, Shrivenham, 1985.

[77] Press, W. H., Teukolsky, S. A., Vetterling, W. T., und Flannery, B. P.: *Numerical Recipes in C.* 2. Auflage. Cambridge University Press 1994.

[78] Reed, R.: Pruning algorithms — A survey. *IEEE Transactions on Neural Networks*, *Bd. 4 [1993]* Nr. 5, S. 740–746.

[79] Riedmiller, M. und Braun, H.: A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *Proceedings of the IEEE International Conference on Neural Networks 1993 (ICNN 93)*, 1993.

[80] Rojas, R.: *Theorie der neuronalen Netze. Eine systematische Einführung.* Springer Verlag, Heidelberg 1993.

[81] Rosenblatt, F.: The perceptron: A perceiving and recognizing automaton. *Report: 85-460-1*, Hrsg. Project PARA, Cornell Aeronautical Laboratory, Ithaca, New York 1957.

[82] Rosenblatt, F.: The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, *Bd. 65* [1958], S. 386–408.

[83] Rumelhart, D. E. und McClelland, J. L.: Learning internal representations by error propagation. In *Parallel Distributed Processing*, Bd. 1. MIT Press, 1986, S. 318–362.

[84] Rumelhart, D. E. und McClelland, J. L.: *Parallel Distributed Processing.* MIT Press 1986.

[85] Sarle, W. S.: Stopped training and other remedies for overfitting. In *Proceedings of the 27th Symposium on the Interface*, 1995.

[86] Schittenkopf, C., Deco, G., und Brauer, W.: Two strategies to avoid overfitting in feedforward networks. *Neural Networks*, *Bd. 10 [1997]* Nr. 3, S. 505–516.

[87] Schmidhuber, J. H.: Adaptive confidence and adaptive curiosity. *Technischer Bericht: FKI-149-91*, Hrsg. Institut für Informatik, Technische Universität, München 1991.

[88] Schmidhuber, J. H.: Curious model-building control systems. In *Proceedings of International Joint Conference on Neural Networks*, Bd. 2. IEEE, Singapore, 1991, S. 1458–1463.

[89] Schmidhuber, J. H.: A fixed size storage $O(n^3)$ time complexity learning algorithm for fully recurrent continually running networks. *Neural Computation*, *Bd. 4 [1992]* Nr. 2, S. 243–248.

[90] Schmidt, R. F., Dudel, J., Jänig, W., und Zimmermann, M.: *Grundriß der Neurophysiologie.* 6. Auflage. Berlin: Springer Verlag 1987.

[91] Schwarz, G.: Estimating the dimension of a model. *The Annals of Statistics*, *Bd. 6* [1978], S. 461–464.

[92] Shewchuk, J. R.: An introduction to the conjugate gradient method without the agonizing pain. *Technical Report: CS-94-125*, Hrsg. Carnegie Mellon University, School of Computer Science 1994.

[93] Stoer, J.: *Einfuehrung in die Numerische Mathematik I.* 5. Auflage. Berlin: Springer Verlag 1989.

[94] Sturm, M.: Simulation einer Gebäudeheizung. *Forschungsberichte Künstliche Intelligenz: FKI-227-98*, Hrsg. Lehrstuhl Prof. Dr. Dr. h.c. W. Brauer, Institut für Informatik, Technische Universität München 1998.

[95] Sturm, M. und Eder, K.: Self-organizing process state detection for on-line adaptation tasks. In Bulsari, A. B., Kallio, S., und Tsaptsinos, D. (Hrsg.), *Solving Engineering Problems with Neural Networks, Proceedings of the International Conference EANN ´96.* Systeemitekniikan seura ry, London, 1996, S. 33–36.

[96] Sturm, M., Eder, K., Brauer, W., und Gonzáles, J. C.: Hybridization of neural and fuzzy systems by a multi agent architecture for motor gearbox control. *Fuzzy Sets and Systems*, *Bd. 89 [1997]* Nr. 3, S. 309–319.

[97] Sturm, M.: Untersuchungen zur Rückgekoppelten Hypothesenbildung in Single Spike Netzen. Diplomarbeit, Institut für Informatik, Lehrstuhl Prof. Dr. Dr. h.c. W. Brauer, Technische Universität München 1993.

[98] Sturm, M.: The MLPNet Class Family. In T.Brychcy/C.Kirchmair/M.Sturm (Hrsg.), *ACON Model Classes, Technical Documentation.* Institut für Informatik, Technische Universität München, 1997.

[99] Sturm, M. und Brychcy, T.: On-Line Prozeßraumkartierung mit ellipsoider Vektorquantisierung zur lokalen Modellbildung. In A.Grauel/W.Becker/F.Belli (Hrsg.), *Fuzzy-Neuro-Systeme ´97 - Computational Intelligence*, Proceedings in Artificial Intelligence. Infix Verlag, St. Augustin, 1997, S. 463–470.

[100] Sturm, M., Brychcy, T., und Kirchmair, C.: AMoC - The ACON Model Classes. *Forschungsberichte Künstliche Intelligenz: FKI-224-97*, Hrsg. Institut für Informatik, Technische Universität München 1997.

[101] Tagscherer, M. und Protzel, P.: Adaptive input-space clustering for continous learning tasks. In Brauer, W. (Hrsg.), *Fuzzy-Neuro Systems ‘98 - Computational Intelligence*, Nr. 7 in Proceedings in artificial intelligence. Infix Verlag, St. Augustin, 1998, S. 352–358.

[102] The MathWorks 1996. *MATLAB Signal Processing Toolbox User´s Guide.* 4. Auflage.

[103] Tikhonov, A. N. und Arsenin, V. Y.: *Solutions of Ill-Posed Problems*. Washington D.C.: V.H. Winston & Sons, John Wiley & Sons 1977. Translation editor Fritz John.

[104] Ungerer, C.: Neuronale Modellierung von Totzeiten in nichtlinearen dynamischen Systemen. Diplomarbeit, Institut für Informatik, Lehrstuhl Prof. Dr. Dr. h.c. W. Brauer, Technische Universität München 1998.

[105] Ungerer, C.: Identifying Time-Delays in Nonlinear Control Systems: An Application of the Adaptive Time-Delay Neural Network. In Mohammadian, M. (Hrsg.), *CIMCA ´99 - Neural Networks & Andvanced Control Strategies*. IOS Press, Netherlands, 1999, S. 99–104.

[106] Ungerer, C., Stübener, D., Kirchmair, C., und Sturm, M.: Supporting traditional controllers of combustion engines by means of neural networks. In Reusch, B. (Hrsg.), *Proceedings of the 6th Fuzzy Days*. Springer-Verlag, 1999, S. 132–141.

[107] Weiß, G. (Hrsg.): *Multiagent Systems. A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA: The MIT Press 1999.

[108] Weigend, A. S., Mangeas, M., und Srivastava, A. N.: Nonlinear gated experts for time-series - discovering regimes and avoiding overfitting. *International Journal of Neural Systems*, *Bd. 6 [1995]* Nr. 4, S. 373–399.

[109] Weigend, A. S., Rumelhart, D. E., und Huberman, B. A.: Generalization by weight-elimination with application to forecasting. In Lippmann, R. P., Moody, J. E., und Touretzky, D. S. (Hrsg.), *Advances in Neural Information Processing Systems*, Bd. 3. Morgan Kaufmann, San Mateo, CA, 1991, S. 875–882.

[110] Williams, R. J. und Zipser, D.: A learning algorithm for continually running fully recurrent neural networks. *Technischer Bericht: ICS Report 8805*, Hrsg. Institute for Cognitive Science, University of California, San Diego, CA 1988.

[111] Williams, R. J. und Zipser, D.: Gradient-based learning algorithms for recurrent connectionist networks. *Technischer Bericht: NU-CCS-90-9*, Hrsg. College of Computer Science, Northeastern University, Boston, MA 1990.

[112] Wolpert, D. H.: On bias plus variance. *Neural Computation*, *Bd. 9 [1997]* Nr. 6, S. 1211–1243.

[113] Wooldridge, M. und Jennings, N.: Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, *Bd. 10(2)* [1995], S. 115–152.

[114] Zeevi, A. J., Meir, R., und Adler, R. J.: Non-linear models for time series using mixtures of autoregressive models. *Journal of Time Series Analysis*. Submitted for publication (1999), Preprint available via `ftp://dumbo.technion.ac.il/pub/PAPERS/mixAR.pdf`.

Index

A

Adaptive Neugier, 50
Adaptive Time Delay Neural Network, 31, 90, 91
Aktionspotential, 23, 24
Anstiegsantwort, 6, 10
ARMA, *siehe* Auto-Regressive-Moving-Average
ATNN, *siehe* Adaptive Time Delay Neural Network
Ausgangsgleichung, 8
Ausgangsgröße, 8
Auto-Regressive-Moving-Average, 30
Axon, 23

B

Backpropagation, 20, 25
Backpropagation Through Time, 28
BFGS-Verfahren, *siehe* Broyden-Fletcher-Goldfarb-Shanno-Verfahren
Bias, 17
Bias-Variance-Problem, 17, 18
Bias-Vektor, 24
Blockdiagramm, 5
Bootstrapping, 19
BPTT, *siehe* Backpropagation Through Time
Broyden-Fletcher-Goldfarb-Shanno-Verfahren, 22
Buchberger-Algorithmus, 60

C

Clustering
  On-Line, 65
Crossvalidation, 19, 74

D

Davidson-Fletcher-Powell-Verfahren, 22
Dendrit, 23
Depolarisation, 23, 24
DFP-Verfahren, *siehe* Davidson-Fletcher-Powell-Verfahren
Dirac-Funktion, 6
DMLP, *siehe* Multi-Layer-Perzeptron, Dynamisch
DRBF, *siehe* Radial-Basis-Function-Netze, Dynamisch
Drift, 10

E

Early Stopping, *siehe* Lernen, Vorzeitiges Abbrechen
Einheitssprung, 6
Ellipsoid
  Abstandsmaß
    Heuristisch, 56
    Heuristisch, Modifiziert, 57
    Komplexitätsreduktion, 61
    Korrekt, 58
  Definition, 54
  Räumliche Adaption, 68
  Zugehörigkeit, 56
Ellipsoide Karte, 54
Elman-Netz, 28
erf(x), 46

F

Fehlerfunktion
  Quadratisch, 20
Feuerfrequenz, 23
Filter, 37
  Ausreißer, 43, 44
  Butterworth, 38, 79
  Chebyshev, 38, 79
  Elliptisch, 38
  Finite-Impuls-Response, 38, 79
  Infinite-Impuls-Response, 38
  Modellbasiert, 39
  Signalsprung, 43, 44
FIR, *siehe* Filter, Finite-Impuls-Response
Flat Minimum Search, 20, 33
FMS, *siehe* Flat Minimum Search
`FOOL`, 74

FOX, 74
Frequenzbereich, 6
Führungsgröße, 8
Fuzzy-Logik, *siehe* Lokale Modellbildung, Modellauswahl, Regelbasis

G

Gauß-Newton-Verfahren, 22
Gebäudeheizung, 91, 101
Generalisierung, *siehe* Modell, Generalisierung
Generalisierungsfehler, 19
Generalized Prediction Error, 19
Gewicht, 23
Gradientenabstieg, 20
  Batch-Verfahren, 21
  Einzelschrittverfahren, 21
  Schrittweitensteuerung, 21, *siehe auch* RProp

H

Hurwitz-Kriterium, 10
Hybride Systeme, 98
Hysterese, 7, 10, 84

I

IIR, *siehe* Filter, Infinite-Impuls-Response
Impulsantwort, 6

J

Jacobi-Matrix, 16, 20
  Adaptive Time Delay Neural Network, 32
  Backpropagation, 25
  Korrektes ellipsodes Abstandsmaß, 60
  Real Time Recurrent Learning, 29
Jordan-Netz, 28

K

Konjugierte Gradienten, 22
Kontextschicht, 28
Korrespondenztabellen, 6
Kostenfunktion, 50

L

Laplace-Transformierte, 6
Learning-Vector-Quantisation, 54
LEMON, 53, 69, 81, 91, 115
  Visualisierung, 54, 83, 88, 92
Lernen, 16
  Allgemeine Modelle, 20
  Ausreißer, 34
  Differenzierbare Modelle, 20
  Extrapolation, 36
  Generalisierung, *siehe* Modell, Generalisierung
  Informationsverlust, 36
  Lokal, 53
  Modellauswahlverfahren, 19
  Rauschen, 34
  Regularisierung, 19
  Signalsprung, 34
  Vorzeitiges Abbrechen, 19
Lernverfahren, *siehe* Lernen
Levenberg-Marquardt-Verfahren, 22, 61
Lokale Modellbildung, 53, 69
  Modellauswahl, 73
    Regelbasis, 74
Lokales Minimum, 20
Long Short-Term Memory, 32
Lose, 7, 10
LSTM, *siehe* Long Short-Term Memory
LVQ, *siehe* Learning-Vector-Quantisation

M

Membranpotential, 24
Mixture of Experts, 97
MLP, *siehe* Multi-Layer-Perzeptron
Model Selection, *siehe* Lernen, Modellauswahlverfahren
Modell, 15
  Adaption, *siehe* Lernen
  Allgemeines, 16
  Black-Box, 16
  Differenzierbar, 16
  Fehlerfunktion, 16
  Generalisierung, 19
  Granularität, 15
  Grey-Box, 15
  Innerer Zustand, 16
  Konfidenz, 65, 72
    Bayes-Modell, 72
    Heuristik, 72
    Neuronaler Schätzer, 72
  Lernen, *siehe* Lernen
  Parametersatz, 16
  Regelungstechnik, *siehe* Regelung, Modellgestützt

White-Box, 15
Modellbildung, 1
Lokal, *siehe* Lokale Modellbildung
On- bzw. Off-Line, 11
Multi-Layer-Perzeptron, 24, 25
Dynamisch, 31
Multiagentensysteme, 96
Multiplikative Synapse, 32

N

Nennerpolynom, 7
Neuron, 23
Neuronaler PID-Regler, 27, 89
Neuronales Netz, 23
Allgemeine Funktionsapproximation, 24
Externer Takt, 27
Interner Zustand, 27
Lernen, 28
Schichten, 23
Vorwärtsgerichtet, 23
Nichtlinearität, 7, 10, 11, 13
Auslöschung, 14
Egalisierung, 14
Normalverteilung, 45
Nyquist-Kriterium, 10

O

OBD, *siehe* Optimal Brain Damage
OBS, *siehe* Optimal Brain Surgeon
OLVQ1, 67, *siehe auch* Learning-Vector-Quantisation
Optimal Brain Damage, 19, 34
Optimal Brain Surgeon, 19, 34
Optimal Experiment Design, *siehe* Optimaler Meßpunkt
Optimaler Meßpunkt, 50
Overfitting, *siehe* Überlernen

P

PID-Regler, 8
Neuronal, *siehe* Neuronaler PI-Regler
PINN, *siehe* Neuronaler PI-Regler
Polstellen, 9
Verdeckt, 9, 10
Prozeß, 5
Prozeßmodell, 11
Invers, 12, 13
Pruning, 19, 34
Pseudoinverse, 40
Explizite Form, 40

Q

Quadrik, 54
Quasi-Newton-Verfahren, 22
QuickProp, 25

R

Radial-Basis-Function-Netze, 26
Dynamisch, 31, 90, 91
RBF, *siehe* Radial-Basis-Function-Netze
Real Time Recurrent Learning, 29
Regeldifferenz, 8
Regelfehler, 5
Regelgröße, 5, 8
Regelkreis, *siehe* Regler
Regelung, 5
Kostenfunktion, 12
Modellgestützt, 11
Modellvorhersage, 12
Vorsteuerung, 12
Zeithorizont, 12
Zielfunktion, *siehe* Regelung, Kostenfunktion
Regler, 8
Regressionsfilter, 42, 77, 79
Frequenzgang, 44
Sprungantwort, 48
Sprungerkennung, 47
Rekurrentes Netz, 27
Lokal rekurrent, 30
Vorstrukturiert, 27
Resampling, 19
RProp, 21
RTRL, *siehe* Real Time Recurrent Learning

S

Schwarz's Bayesian Criterion, 19
Sigmoidfunktion, 24
Verschwindender Gradient, 25
Simulated Annealing, 21
Singuläre-Werte-Zerlegung, 43, 64
Sprungantwort, 6, 10
Sprungerkennung, 44, *siehe auch* Regressionsfilter, Sprungerkennung
Stabilität, 9
Stellgröße, 5, 8
Steuerung, 5
Störgröße, 5, 8

Strafterm, 12, 19, 20
Strecke, 5
Synapse, 23

T
TDNN, *siehe* Time Delay Neural Network
Teacher Forcing, 30
Testmenge, 19
Time Delay Neural Network, 30
Totzeit, 11, 13, 32
Trainingsmenge, 17, 19

U
Überlernen, 18, 19, *siehe auch* Lernen, Bias-Variance-Problem
Übertragungsfunktion, 6
Übertragungsglied, 6
Underfitting, *siehe* Unterlernen
Unterlernen, 18, *siehe auch* Lernen, Bias-Variance-Problem

V
Validierungsmenge, 19
Variance, 17
Visualisierung, *siehe* LEMON, Visualisierung
Vorlast, 7, 10
Voronoi-Parkettierung, 54, 62
Vorsteuerung, 12

W
Weight-Decay, 19, 33

Z
Zählerpolynom, 7
Zeitbereich, 6
Zeitfunktion, 6
Zustand, 8
Zustandserkennung, 66
Zustandsgleichung, 8
Zustandsraum, 8

Modellbildung in der Regelungstechnik, also die möglichst gute Approximation eines technischen Prozesses durch ein mathematisches bzw. physikalisches Modell, stellt eine große Herausforderung dar. Speziell die On-Line-Fähigkeit der Modellierung ist meist nicht erfüllbar.

In dieser Arbeit wird eine vereinheitlichte Sichtweise gängiger Modellbildungsverfahren gegeben, die insbesondere das Feld der neuronalen Netze umfaßt.

Die Forderung „On-Line-Fähigkeit“ und „Modellierung nichtlinearer Systeme“ führt zum Algorithmus LEMON (**L**ocal **E**llipsoidal **Mo**del **N**etwork). Dieser basiert auf der Verwendung einer ellipsoiden Kartierungsmethode kombiniert mit einer lokalen Modellbildung, die eine *automatische* Anpassung der lokalen Modellkomplexität an die Prozeßkomplexität erlaubt.

Zur Behandlung verrauschter Daten mit Ausreißern und Sprüngen im Nutzanteil des Eingabesignals, wird ein Regressionsfilter entworfen, der mit Hilfe einer statistischen Sprungerkennung einzelne Ausreißer von echten Sprüngen der Originaldynamik trennt.

Adaptive CONTROL

Diese Arbeit wurde vom Bundesministerium für Bildung, Wissenschaft, Forschung und Technologie im Rahmen des Projektes ACON (FKZ 01IN510C8) gefördert.