Entwicklung und Implementierung eines Verfahrens zur dynamischen Optimierung von Kraftwerksfahrweisen und Anwendung in kommerziellen Simulationsprogrammen

Development and implementation of a method for the dynamic optimization of power plant operation modes and application in commercial simulation programs

Master-Thesis von Robert Schiemann aus Leipzig
Prof. Dr.-Ing. Bernd Epple
Dipl.-Ing. Ralf Starkloff
Oktober 2014

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich 16: Maschinenbau
Institut für Energiesysteme und Energietechnik

Entwicklung und Implementierung eines Verfahrens zur dynamischen Optimierung von Kraftwerksfahrweisen und Anwendung in kommerziellen Simulationsprogrammen
Development and implementation of a method for the dynamic optimization of power plant operation modes and application in commercial simulation programs

Vorgelegte Master-Thesis von Robert Schiemann aus Leipzig
Prof. Dr.-Ing. Bernd Epple
Dipl.-Ing. Ralf Starkloff

1. Gutachten: Prof. Dr. Bernd Epple
2. Gutachten: Dipl.-Ing. Ralf Starkloff

Tag der Einreichung:

Erklärung zur Master-Thesis

Hiermit versichere ich, die vorliegende Master-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 9. September 2014

(Robert Schiemann)

Aufgabenstellung zur Masterthesis M-192-16

Für:	Herrn Robert Schiemann
Matr.-Nr.:	1570982
Institut:	Energiesysteme und Energietechnik
Betreuer:	Prof. Dr.-Ing. B. Epple
	Dipl.-Ing R. Starkloff

TECHNISCHE UNIVERSITÄT DARMSTADT

Titel

Entwicklung und Implementierung eines Verfahrens zur dynamischen Optimierung von Kraftwerksfahrweisen und Anwendung in kommerziellen Simulationsprogrammen
Development and implementation of a method for the dynamic optimization of power plant operation modes and application in commercial simulation programs

Hintergrund

Kohlekraftwerke sind in der näheren Zukunft zur Abdeckung des Grundlastbedarfs unverzichtbar. Ihr im Vergleich zu anderen Kraftwerken träges An- und Abfahrverhalten rückt durch steigende Flexibilitätsanforderungen einer Kraftwerksanlage gegenüber der Stromnachfrage immer mehr in den Mittelpunkt des Interesses. Zügiges und wirtschaftliches An- und Abfahren eines Kohlekraftwerkes sowie ein gutes Lastwechselverhalten sind für ein modernes Kohlekraftwerk essenziell.
Jedoch ist im Planungsstadium eines Kraftwerkes eine exakte Vorhersage seines dynamischen Verhaltens nur schwer möglich. Zudem sind Versuche zur Optimierung dynamischer Prozesse eines Kohlekraftwerkes teuer und aus Sicherheitsgründen nicht immer machbar. Die Simulation dynamischer Prozesse ermöglicht Einblicke in die Vorgänge eines Kraftwerkes mit (im Vergleich zu Versuchen) geringem Aufwand und ohne Sicherheitsrisiko. Dies und die Tatsache immer besser werdender Simulationsmöglichkeiten machen die numerische Simulation zu einem wichtigen Bestandteil der Kraftwerksplanung und Kraftwerksentwicklung.

Zielstellung

Ziel der Arbeit ist es existierende Optimierungsverfahren auf instationäre Vorgänge im Kraftwerksbetrieb anzuwenden. Hierbei sollen verschiedene Optimierungsstrategien untersucht werden, wobei nach ein geeignetes ausgewählt werden. Dieses soll in einem numerischen Simulationsprogramm angewendet werden um die Betriebsweise der Anlage nach verschiedenen Kriterien zu optimieren.

Arbeitsschritte

1. Einarbeitung in die Optimierungsthematik
2. Ermitteln und Vergleichen unterschiedlicher Optimierungsstrategien
3. Entwickeln von geeigneten Verfahrensauswahlkriterien zur Anwendung in kommerziellen Simulationsprogrammen
4. Auswahl eines Verfahrens
5. Implementierung des Verfahrens
6. Anwenden des Verfahrens auf unterschiedlich komplexe Beispielsysteme
    a. Prinzipielle Überprüfung des Optimierungsverfahren in einfachen Prozessen
    b. Optimierung von Teilsystemen in Kraftwerksmodellen
    c. Anwendung in validierten Großkraftwerksmodellen bei Laständerung
7. Dokumentation der Arbeit

Prof. Dr.-Ing Bernd Epple

Zusammenfassung

Nicht zuletzt aufgrund der gegenwärtig steigenden Integration regenerativer Energieträger sehen sich konventionelle Stromerzeugungsanlagen hinsichtlich ihrer Flexibilität und Wirtschaftlichkeit zunehmend höheren Anforderungen gegenübergestellt. Neben vielen technischen Detaillösungen ist vor allem eine optimale Prozessführung von hoher Bedeutung, um diesen Anforderungen nachzukommen. Derartige bestmögliche Prozesssteuerungen können unter verschiedensten Beschränkungen und Bewertungsgesichtspunkten durch die Lösung eines Optimalsteuerungsproblems berechnet werden, wofür jedoch ein explizites Systemmodell vorliegen muss. Dieses ist jedoch in vielen praktischen Fällen – weder bei physischen Anlagen, noch bei den meisten Simulationsumgebungen – nicht gegeben. Gegenstand der vorliegenden Arbeit war es daher, ein Verfahren zu entwickeln, das diese Lücke schließt und somit eine Anwendung der Theorie der optimalen Steuerungen auf Simulationsumgebungen, aber prinzipiell auch physische Systeme erlaubt.

Zu Lösung dieser Aufgabe wurde eine Vorgehensstrategie entwickelt, die zunächst das Ermitteln eines mathematischen Ersatzmodells beinhaltet. Da dieses möglichst viele unterschiedliche Systeme nachbilden können soll, wurde auf die universale Struktur der neuronalen Netze zurückgegriffen. Mit diesen als Ersatzmodell ist dann eine numerische Berechnung der optimalen Steuerungen möglich. Im Rahmen dieser Arbeit wurden unterschiedliche Methoden zur Erzeugung eines Ersatzmodells und zur Optimierung ausgearbeitet und miteinander verglichen.

Nach der Entwicklung des Gesamtverfahrens wurde dieses anhand dreier Anwendungsbeispiele mit steigender Komplexität getestet. Auf diese Weise konnte beispielsweise für eine reales Gas-und-Dampf-Kraftwerk in Malaysia eine mögliche Beschleunigung eines Warmstarts und die dazugehörige Steuerung identifiziert werden. In einem anderen Testfall wurde das Lastwechselverhalten eines Steinkohleblocks in Deutschland untersucht. Hierbei ergab sich, dass die in der Anlage implementierten Regelungsschaltungen einen nahezu optimalen Lastwechsel ermöglichen. Darüber hinaus wurde der mögliche Zeitvorteil eines Lastwechsels bei höheren zulässigen Änderungen der Frischdampftemperatur diskutiert. Insgesamt lieferte das Gesamtverfahren unter Beachtung einiger wichtiger Bedingungen gute Ergebnisse sowohl im Schritt der Systemidentifikation als auch in dem der Optimierung.

Inhaltsverzeichnis

Symbol- und Abkürzungsverzeichnis

Römische Symbole

$\tilde{g}$	Gleichungsnebenbedingungen an ein NLP
$\tilde{h}$	Ungleichungsnebenbedingungen an ein NLP
c_p	spezifische Wärmekapazität bei konstantem Druck in kJ/kg
d	Verzögerungswert (delay)
f	Allgemeine Funktion/Systemfunktion
g	Beschränkungen von Systemzuständen oder Steuerungen
H	Ausgangswert eines Neurons in der versteckten Schicht, Hamiltonfunktion
I	Einheitsmatrix
J	Jacobi-Matrix
L	Lagrange-Term/Lipschitzkonstante
N	Anzahl der Datenpunkte in einem Trainingsdatensatz
P	Leistung in W
p	Druck in Pa, Optimierungsparameter in einem NLP
r	Radius in m
T	Temperatur in °C
U	Menge der zulässigen Steuerungen
u	Steuerung
W	Gewichte in einem neuronalen Netz (Matrix)
w	Gewicht in einem neuronalen Netz (Skalar)
x	(innerer) Systemzustand
y	Systemausgangsgröße

Griechische Symbole

α	Auftretende Konstante bei der Umwanldung von NARX- in zeitstetige Netze
β	Auftretende Konstante bei der Umwanldung von NARX- in zeitstetige Netze
δ	Auftretende Konstante bei der Umwanldung von NARX- in zeitstetige Netze
ϵ	Residuum/Vorhersagefehler

η	Parametervektor eines neuronalen Netzwerks, Multiplikatorfunktion
γ	Auftretende Konstante bei der Umwanldung von NARX- in zeitstetige Netze
κ	Auftretende Konstante bei der Umwanldung von NARX- in zeitstetige Netze
λ	Lagrange-Multiplikator
ω	Gewichtung
ϕ	Mayer-Term
σ	sigmoide Aktivierungsfunktion/thermische Spannung in MPa
τ	Zeitkonstante in Sekunden
φ	Kostenfunktion eines NLPs
ξ	Modell-Parameter

Abkürzungen, Indizes und obere Indizes

$\hat{}$	Ersatzmodell
0	Auslegungs-/Nominalwert einer Größe
a	außen
ASME	American Society of Mechanical Engineers
avg	Gemittelter Wert (average)
BPVC	Boiler and Pressure Vessel Code
CNCO	Kontinuierliches Netz mit kontinuierlicher Optimierung
Cov	Kovarianz
CTRNN	Continuous time recurrent neural network
DDoE	Dynamic Design of Experiments
dim	Dimension/Anzahl der Elemente eines Vektors
DNCO	Diskretes Netz mit kontinuierlicher Optimierung
DNDO	Diskretes Netz mit diskreter Optimierung
DoE	Design of Experiments
DTRNN	Discrete time recurrent neural network
EEG	Erneuerbare-Energien-Gesetz
el	elektrisch
EM	Ersatzmodell
f	final, Endzustand
FD	Frischdampf

GT	Gasturbine
GuD	Gas- und Dampfkraftwerk
H	Hidden layer (versteckte Schicht)
hXdY	Abkürzung für ein neuronales Netz mit $n_h = X$ und $d_u = d_y = Y$
HO	von der versteckten zur Ausgangsschicht
i	innen
IH	von der Eingangs- zur versteckten Schicht
inp	Eingangsgröße in ein Netz
MLP	Multi-layer perceptron (Mehrschicht-Perzeptron)
NARX	Nonlinear Autoregressive exogenous neural network
NLP	Nonlinear Programming Problem
O	Output layer (Ausgangsschicht)
RG	Rauchgas
s	Schaltpunkt
SPAT	Speisepumpenantriebsturbine
SQP	Sequentielle quadratische Programmierung
th	thermisch
TRD	Technische Regeln für Dampfkessel
Zuteil	Brennstoffzuteiler

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Problembeschreibung

Im Zuge gegenwärtiger Bestrebungen zur Erhöhung des Anteils regenerativ erzeugten Stroms ist insbesondere in Deutschland, aber auch vielen anderen europäischen Ländern ein Wandel in der Struktur des Kraftwerksparks im Gange. Dies liegt in erster Linie an der volatilen Natur der regenerativen Energieerzeugung. Das betrifft im Besonderen Windkraft-, solarthermische und Photovoltaik-Anlagen, deren Stromerzeugungsvermögen stark abhängig von Wettereinflüssen ist und sich somit binnen sehr kurzer Zeiträume stark verändern kann. Diese Schwankungen können zum gegenwärtigen Zeitpunkt nicht immer zuverlässig vorhergesagt werden und wirken somit direkt auf das Verbundnetz ein (Scheibner 2014). Hinzu kommt, dass im bundesdeutschen Erneuerbare-Energien-Gesetz (EEG) seit dem Jahr 2000 unter anderem der Einspeisevorrang regenerativ erzeugten Stroms festgelegt ist, was die Stabilität der Stromnetze zusätzlich gefährdet. Dies ist jedoch nicht nur ein Problem Deutschlands, da in Europa großflächige Verbundnetze existieren und zudem europa- und weltweit viele Staaten ähnliche Regulierungen verabschiedet haben, die sich am EEG orientieren.

Aufgrund des Vorrangs und der gleichzeitigen Volatilität der Einspeisung regenerativ erzeugten Stroms ist es also dem konventionellen Kraftwerkspark aufgebürdet, die Differenz zwischen der Lastanforderung und dem regenerativ erzeugten Strom, die sogenannte Residuallast, möglichst so zu decken, dass weder Strommangel noch Stromüberschuss im Netz entstehen. Diese Anforderung ist von eminenter Wichtigkeit für die Stabilität des Stromnetzes und drückt sich in einer möglichst stabilen Frequenz der Netzspannung aus, deren nomineller Wert in Europa bei 50 Hz liegt.

Wegen der direkten Wetterabhängigkeit des regenerativ erzeugten Stroms kann die Residuallast innerhalb weniger Minuten oftmals um viele Gigawatt variieren, was eine große Herausforderung an den konventionellen Kraftwerkspark stellt (Scheibner 2014). Dieser muss die Einspeiseschwankungen wie bereits erwähnt auffangen und ausregeln, was in häufigen und hohen Lastwechselanforderungen für die jeweiligen Anlagen resultiert.

Ein solches Lastregime, in dem in kurzen Zeitspannen mitunter große Lastwechsel vollzogen werden müssen, ist bei vielen thermischen Kraftwerken nicht auslegungsgemäß. Dies trifft besonders auf Braunkohle-, Steinkohle- und Kernkraftwerte zu, welche ursprünglich für den Grund- und Mittellastbetrieb konzipiert wurden. In der Folge werden diese Anlagen bei den Lastwechseln erhöhten Belastungen ausgesetzt, welche die Lebensdauer wichtiger und hochbeanspruchter Anlagenteile zusätzlich reduzieren. Hierdurch und durch große Einbußen in der Anlagenauslastung infolge des Einspeisevorrangs regenerativ erzeugten Stroms ist die Wirtschaftlichkeit derartiger Anlagen gegenwärtig grundsätzlich in Frage gestellt. Daher sind neue Betriebsstrategien und technische Maßnahmen notwendig, um Alt- und Neuanlagen möglichst flexibel auszuführen und zu betreiben.

Dies ist eine der zentralen Herausforderungen der Energieerzeugung der nächsten Jahre und Jahrzehnte in einer Gesellschaft, die einen Paradigmenwechsel zu einer möglichst regenerativen Energieversorgung zu vollziehen versucht. Um der Anforderung der notwendigen Flexibilisierung des konventionellen Kraftwerksparks gerecht zu werden, wurden von technischer Seite bereits einige Maßnahmen vorgeschlagen und implementiert, die einerseits konstruktive Detaillösungen an dynamisch trägen Bauteilen und andererseits verbesserte Betriebsstrategien umfassen. Beispiele für derartige Maßnahmen sind:

- Die konstruktive Optimierung dickwandiger Komponenten, wie etwa Trommeln, Sammler und die Turbinenrotoren. Diese sind aufgrund ihrer Geometrie und hohen thermischen Belastung starken thermomechanischen Spannungen ausgesetzt, welche in der Regel ein gewisses Materiallimit nicht

überschreiten dürfen und somit einen begrenzenden Faktor für die Geschwindigkeit dynamischer Vorgänge darstellen (Schmidt u. Schuele 2013)

- Der Abbau von Konservativitäten bei der Auslegung von Druckteilen oder den Lebensdauerbewertungen. Vergleiche verschiedener Auslegungsregeln, etwa nach DIN 12952, den technischen Regeln für Dampfkessel (TRD) und dem ASME Boiler and Pressure Vessel Code VIII (BPVC) haben gezeigt, dass je nach zugrunde gelegter Norm deutliche Einsparungen in der Wandstärke druckführender Bauteile erreicht werden kann, was deren thermische Trägheit und Schädigungspotenzial stark reduziert (Grammenoudis u. Weber 2011, Schmidt u. Schuele 2013).

- Das Einführen von Schwenkbrennern und dem Einmühlenbetrieb (Schmidt u. Schuele 2013, Heinzel u. a. 2012) zur Absenkung der Mindestlast. Dies ist für die Wirtschaftlichkeit einer Anlage besonders wichtig, um selbst in Zeiten niedriger Residuallasten immerhin am Netz bleiben zu können.

- Das Konzept der indirekten Feuerung. Durch eine Zwischenbunkerung vorgemahlenen Brennstoffs kann wesentlich schneller auf eine geänderte Lastanforderung reagiert werden, da Kohlemühlen eine hohe Trägheit aufweisen und somit kritisch für Lastwechselgeschwindigkeiten sind (Schmidt u. Schuele 2013).

- Flexible Anlagen- oder Betriebskonzepte. Dies umfasst beispielsweise den Kondensatstopp oder die Umfahrung der Hochdruck-Vorwärmer, die zum Ziel haben, durch das Schließen von Anzapfungen an der Dampfturbine kurzfristig eine erhöhte Leistung zu erzeugen (Zehtner 2009, Starkloff u. a. 2013). Diese Primärregelungsmaßnahme ist besonders in Zeiten zunehmenden regenerativ erzeugten Stroms wichtig für die Stabilität der Netzfrequenz.

Neben diesen technischen Detaillösungen ist darüber hinaus von besonderem Interesse, die eigentliche Fahrweise der Anlage genau in der Art zu steuern, dass diese in einem gewissen Sinne optimal ist, jedoch gewisse Beschränkungen einhält. Man spricht bei dieser Problemstellung von einem *Optimalsteuerungsproblem*. Ein naheliegendes Beispiel für ein solches Problem ist der möglichst schnelle Lastwechsel einer Anlage in einen definierten Zielzustand. Neben der Forderung von möglichst schnellen Aktionen können aber auch andere Gütekriterien wie etwa der Brennstoffverbrauch während der Aktion, der Steueraufwand, die kumulierte Stromerzeugung oder eine gewichtete Kombination dieser Gütekriterien von Interesse sein, was vor allem für praktische Fälle einleuchtend ist. Von großer Bedeutung ist bei der steuerungstechnischen Optimierung hinsichtlich der eben genannten Gütekriterien, die Anlage möglichst schonend zu betreiben. Dazu gehört insbesondere das Mäßigen der Laständerungsgeschwindigkeiten, sodass die in dickwandigen Bauteilen entstehenden thermischen Spannungen nicht zu groß werden. Diese Nebenbedingung widerspricht direkt dem Wunsch nach schnellen Lastwechseln, sodass hier immer ein gewisser Kompromiss zu suchen ist. Auch andere Nebenbedingungen können für die Prozesse von Bedeutung sein, wie beispielsweise Beschränkungen in der Steuerung. Dies können etwa etwa maximale Stellgeschwindigkeiten von Ventilen oder Zeitkonstanten von Pumpen und anderen Hilfseinrichtungen sein.

Diese Anforderungen und Nebenbedingungen zusammen bilden eine mathematische Formulierung des Optimalsteuerungsproblems, welche im Allgemeinen sehr komplex und meist nur numerisch lösbar sind. Zur Formulierung und Lösung derartiger Probleme ist allerdings ein explizites mathematisches Modell des zu optimierenden Prozesses notwendig, welches in der Praxis oftmals nicht oder nur eingeschränkt vorliegt. Dies ist besonders für eine simulationsgestützte Optimierung der Fall, bei der komplexe Anlagen in hochentwickelter, kommerzieller Simulationssoftware modelliert und simuliert werden, jedoch dem Nutzer gegenüber als Black-Box fungieren.

Gegenstand dieser Arbeit ist die Kopplung geeigneter Optimierungsmethoden mit kommerziellen Simulationssystemen, um die vorstehend genannte Klasse von Optimalsteuerungsproblemen unter der Benutzung von Simulationsmodellen lösen zu können.

Hierzu ist die Arbeit in zwei Teile gegliedert. Nach dieser Einleitung wird im ersten Teil, welcher sich mit der Entwicklung eines entsprechenden simulationsgestützten Optimierungsverfahrens beschäftigt, in

Kapitel 2 ein grundlegendes Verfahrenskonzept erarbeitet und diskutiert. Kapitel 3 ist der theoretischen Aufarbeitung wichtiger mathematischer Grundlagen, die von Bedeutung für das Verfahrenskonzept sind, gewidmet. Die konkrete Struktur und Implementierung des Verfahrens sind dann im Kapitel 4 dargelegt. Im zweiten Teil der vorliegenden Arbeit soll sodann die Erprobung und Validierung des konzeptionierten Verfahrens im Mittelpunkt stehen. Hierzu werden in den Kapiteln 6 bis 8 zunehmend komplexere und praxisnähere Beispielprobleme einer Optimierung unterzogen und die Qualität der erzielten Lösungen diskutiert. Zum Abschluss dieser Arbeit wird in den Kapiteln 9 und 10 ein Fazit gezogen und ein Ausblick über mögliche zukünftige Arbeiten zu diesem Thema gegeben.

1.2 Stand der Technik

Neben den Maßnahmen, die schon in vorherigen Abschnitt 1.1 genannt wurden, haben in der Praxis bereits einige optimierende Maßnahmen Einzug in den Kraftwerksbetrieb gehalten.

So wird für gewöhnlich ein Monitoring von Wandtemperaturen dickwandiger Bauteile durchgeführt, um eine Schätzung der vorliegenden thermischen Spannungen zu erlauben. Diese werden dann entsprechend in den Regelungen berücksichtigt. Dabei ergibt sich allerdings das Problem, dass aus Gründen der Strukturintegrität die Wandinnentemperatur nicht direkt gemessen werden kann, was die entsprechenden Messungen verzögert und ungenau macht. Aus diesem Grund wurden in der Literatur bereits Methoden vorgeschlagen, in denen Messungen der Fluidtemperatur und der Wandaußentemperatur in Kombination mit mathematischen Modellen die thermischen Spannungen sehr viel genauer vorhersagen können. Ein hierauf basierendes Verfahren wurde im Block 8 des Großkraftwerks Mannheim implementiert, wodurch Brennstoffeinsparungen von 20 bis 50% im Bereich des Möglichen scheinen (Lausterer 1997).

Darüber hinaus werden oftmals sogenannte linear-quadratische (LQ-)Regler genutzt, die optimal Störungen ausregeln oder Abweichungen zu Führungsgrößen vermindern können. Diese besondere Art von Reglern existiert jedoch nur für lineare Systeme und eine bestimmte Klasse von Gütekriterien, was in der Praxis jedoch häufig gegeben ist (Papageorgiou 2006, Kirk 1970). Das ist insbesondere dann der Fall, wenn Abweichungen gewisser Größen von einem Sollwert ausgeregelt werden sollen, weil diese meist klein genug sind um das eigentlich nichtlineare System ohne allzu große Fehler linearisieren zu können.

Neben diesen sehr praxisorientierten Maßnahmen wurden in der Literatur bereits einige Ansätze zur modellbasierten Optimierung von Kraftwerksfahrweisen vorgeschlagen. So wurde etwa ein einfaches, abstrahiertes Modell eines Abhitzedampferzeugers mit der Modellierungssprache Modelica abgebildet und mit der Open-Source-Software JModelica.org hinsichtlich eines schnellen Anfahrprozesses optimiert (Casella u. a. 2011a). Etwas weiter gehen Ansätze, ein komplettes, nichtlineares Modelica-Modell eines kombinierten Gas- und Dampfkraftwerkes zu formulieren und um ausgewählte Betriebspunkte zu linearisieren, um eine einfache und effiziente Optimierung der Steuergrößen zu ermöglichen (Casella u. a. 2011b) oder direkt mit einem nichtlinearen Modell zu optimieren (Albanesi u. a. 2006, Lind u. Sällberg 2012). Auch für einen recht detailliert modellierten Dampferzeuger wurde bereits eine modellgestützte Anfahroptimierung vorgenommen, wodurch große Einsparungen in der Anfahrzeit und dem Brennstoffverbrauch erzielt werden konnten (Krüger u. a. 2001).

Trotz diverser Einzelerfolge haben all diese Beispiele jedoch keinen universalen Charakter. In vielen dieser Fälle waren die Systemmodelle einfach gehalten, um eine effiziente numerische Lösung des Optimierungsproblems zu ermöglichen. Hinzu kommt, dass die Modelle meist nur für eine bestimmte Anlage oder Systemkonfiguration entwickelt wurden und somit ein funktionierendes Optimierungsverfahren nur eingeschränkt auf andere Modelle anwendbar ist. Eine generische Modellierung, wie sie zum Beispiel mit der Modellierungssprache Modelica und einer entsprechenden Simulationsumgebung, wie etwa Dymola, ist aus diesem Grund grundsätzlich ein besserer Ansatz. Allerdings können die Modelle bei realistischer Modellierung so kompliziert sein, dass das Optimierungsproblem numerisch nicht oder nur sehr schwer gelöst werden kann. Die Wurzel dieses Problems ist die Tatsache, dass für eine Lösung des Optimalsteuerungsproblems jede einzelne Modellgleichung des Modells sowie deren Ableitungen bis hin zur zweiten Ordnung mit einbezogen werden müssen (Kirk 1970). Für eine direkte Optimierung sind

hochwertige, realistische Modelle also zu komplex. Wünschenswert wäre demnach ein ganzheitliches Verfahren zur Optimierung der Steuerung von Kraftwerksprozessen, das auch komplexe Systeme behandeln kann. Die Entwicklung eines derartigen Verfahrens ist Gegenstand der vorliegenden Arbeit und wird im folgenden Teil I dieser Arbeit konzeptioniert und ausformuliert.

Teil I.

Entwicklung eines simulationsgestützten Optimierungsverfahrens

2 Konzeptionierung des Verfahrens

2.1 Anforderungen an das Verfahren

Vor der Konzeptionierung des simulationsgestützten Optimierungsverfahrens sollen hier zunächst einige Anforderungen an das Verfahren formuliert werden, die dessen Umsetzbarkeit, Effizienz und Vielseitigkeit sichern sollen.

Allgemeinheit Das verwendete Verfahren soll möglichst unabhängig vom zu behandelnden physikalischen System sein. Nichtlineare oder nicht eingangs-affine Systeme sollten ebenso wenig ein Problem darstellen wie nichtautonome Problemstellungen. Darüber hinaus ist eine Unabhängigkeit von der verwendeten Simulationsumgebung gefordert, sodass das Verfahren schlussendlich mit einer breiten Auswahl an kommerziellen Simulationsprogrammen die dynamische Optimierung durchführbar macht. Darüber hinaus wäre auch eine Anwendung des Verfahrens auf physikalische Systeme wünschenswert, die nicht als Simulationsmodell, sondern als reales System, vorliegen.

Flexibilität Besonders in der Formulierung eines Optimierungsproblems können aus mathematischer Sicht viele Varianten auftreten. Einerseits können dies, wie in Kapitel 1 bereits angedeutet, verschiedene Gütekriterien wie beispielsweise Zeit- oder Energieminimalität sein. Andererseits kann die Formulierung von Nebenbedingungen, unter denen optimiert werden soll, verschieden komplex und realitätsnah erfolgen. So könnte beispielsweise eine Beschränkung für eine thermische Spannung nicht zu allen Zeiten auf einem festen Wert stehen, sondern abhängig vom Innendruck des entsprechenden Druckbehälters sein, um hohe thermische Beanspruchungen bei hohen mechanischen Beanspruchungen zu vermeiden.

Genauigkeit Eine wichtige Anforderung ist die Genauigkeit des im Rahmen des Verfahrens berechneten Optimierungsergebnisses. Diese sollte natürlich – bei vertretbarem Rechenaufwand – hoch genug sein, um aus praktischer Sicht zuverlässige Ergebnisse zu erbringen.

Effizienz, Robustheit und Zuverlässigkeit Um ein effizientes Verfahren zu erhalten, sollten, soweit möglich, bestehende Optimierungsverfahren nutzbar sein. In der klassischen Theorie der optimalen Steuerungen, auf die in Abschnitt 3.2 noch detailliert eingegangen wird, sind eine Fülle an Lösungsstrategien von optimalen Steuerungsproblemen entwickelt worden. Um diese nutzen zu können, ist es notwendig, die Optimierungsprobleme in einer gewissen Standardform zu stellen, was aber in den meisten Fällen intuitiv erfolgen kann. Darüber hinaus sollten bereits existente Implementierungen von Verfahren genutzt werden, um eine hohe Effizienz, Fehlerfreiheit, Robustheit und Zuverlässigkeit zu erreichen.

2.2 Skizzierung des Verfahrens

In diesem Abschnitt soll nun die Erarbeitung eines grundlegenden Verfahrenskonzeptes erfolgen, um in den nachfolgenden Kapiteln die entsprechenden Grundlagen zu behandeln.

Die allgemeine Problemstellung, wie sie im einleitenden Kapitel 1 verbal formuliert worden ist, lautet[1]:

$$\min_{u(t)} \quad J[u(t)] \tag{2.1a}$$

$$\text{u.B.v.} \quad \dot{x}(t) = f(x(t), u(t), t) \tag{2.1b}$$

$$g(x(t)\,, u(t), t) \geq 0 \tag{2.1c}$$

Hierin ist $J[u(t)]$ ein sogenanntes *Gütefunktional*, das von der Steuerung $u(t)$ abhängt. Dieses gilt es im Optimierungsprozess zu minimieren. Als Beschränkungen an die zu ermittelnde optimale Lösung treten an dieser Stelle die Systemdynamik $f(x(t), u(t), t)$ und etwaige weitere Ungleichungsbeschränkungen $g(x(t), u(t), t)$ an die Zustände $x(t)$ oder die Steuerungen $u(t)$ auf. Zusätzlich müssen Anfangs- und eventuell Endbedingungen gegeben sein, um eine wohldefinierte Lösung der Differentialgleichungen zu erhalten.

Zur Lösung derartig gestellter Probleme liefert die Theorie der optimalen Steuerungen die entsprechenden Vorgehen und Verfahren. Hierzu müssen allerdings sämtliche Problemfunktionen explizit gegeben sein. Dies mag bei theoretischen Untersuchungen keine allzu große Schwierigkeit bereiten, doch im praktischen Fall ist besonders die Systemfunktion f in aller Regel in wesentlichen Teilen unbekannt. Das liegt weniger daran, dass keine grundsätzliche physikalische Kenntnis der Prozesse vorhanden ist, sondern dass in der Systemfunktion f gegenüber der Realität erhebliche Parameterunsicherheiten auftreten können. Ein anderes, oft auftretendes Problem ist die mangelnde Güte von viel benutzten Teilmodellen, etwa semi-empirische Korrelationen zur Bestimmung von Wärmeübergangskoeffizienten und Druckverlusten.

Liegt die Systemfunktion nicht vor, so ist immerhin eine modellfreie Optimierung denkbar. Hierzu könnte man beispielsweise die Steuerung $u(t)$ in irgendeiner Weise parametrisieren und die dadurch eingeführten Parameter direkt am Originalsystem, das als Black-Box fungieren würde, optimieren. In diesem Fall hätte man es – im Gegensatz zum Optimalsteuerungsproblem nach den Gleichungen (2.1a) bis (2.1c) – mit einem statischen Optimierungsproblem zu tun, das nur relativ wenig zu optimierende Parameter aufweist. Hierfür steht eine Vielzahl von Verfahren bereit, wie etwa genetische Algorithmen (Mitchell 1996) oder das bekannte Simplex-Verfahren nach Nelder und Mead (Nelder u. Mead 1965).

Bei der Parametrisierung der Steuerung $u(t)$ kann natürlich auch schon physikalische Vorkenntnis über die optimalen Lösung einfließen, wodurch die Anzahl der zu optimierenden Parameter stark reduziert werden kann. In Verbindung mit einer geeigneten Versuchsplanung kann so der Aufwand für die Optimierung drastisch reduziert werden. Diese Methodik wird in Analogie zum klassischen experimentorientierten *Design of Experiments* (DoE) auch häufig *Dynamic Design of Experiments* (DDoE) genannt (Georgakis 2012).

Ein großes Problem bei derartigen modellfreien Verfahren ist die Notwendigkeit sehr vieler Güteauswertungen eines bestimmten Parametersatzes. Es ist intuitiv einleuchtend, dass die Anzahl der Auswertungen bei modellfreien Methoden größer ist als bei den modellbasierten Verfahren (Papageorgiou 2006). Dies macht den modellfreien Ansatz kaum noch praktikabel, da jede einzelne Güteauswertung die komplette Simulation des Systems der mit den Parametern beschriebenen Steuerung $u(t)$ und anschließende Auswertung des Gütekriteriums erfordert. Je nach Systemkomplexität muss dies mehrere hundert- bis tausendmal geschehen. Heutige Simulationsprogramme, insbesondere jene im Kraftwerks- und verfahrenstechnischen Bereich, sind jedoch so hoch entwickelt und komplex, dass eine Simulation einer größeren Anlage lediglich unter Echtzeit ablaufen kann. Dadurch wird deutlich, dass ein modellfreier Optimierungsansatz einen äußerst hohen Zeitaufwand erfordert, sodass dieser Ansatz nicht weiter verfolgt wird. In diesem Sinne sind heutige kommerzielle Simulationsprogramme aus Sicht der Optimierung nicht Black-Box-fähig. Diesen Ausführungen entsprechend ist also eine Optimierung mit einem Systemmodell f wünschenswert. Zu beantworten ist nun die Frage, wie ein solches Modell erhalten werden kann.

[1] die Notation u.B.v. bedeutet „unter der Beschränkung von".

Bei physikalischen Systemen, die in einer Simulationsumgebung abgebildet wurden, ist zwar die Systemfunktion f implizit im Programm vorhanden, aber selten von außen einsehbar. Dies ist beispielsweise in den prominenten Programmen APROS und ASPEN PLUS DYNAMICS der Fall, wo der Nutzer keine Möglichkeit eines Zugriffs auf die interne Berechnungsstruktur hat. Eine Ausnahme bildet hier zum Beispiel das Programmpaket DYMOLA, welches prinzipbedingt ein vom Nutzer zu definierendes und somit einsehbares Systemmodell bereithält, das in expliziter Gleichungsform vorliegt.

Unabhängig davon, ob die Systemfunktion explizit vorliegt oder nicht, wäre diese für die allermeisten praktischen Fälle ohnehin viel zu komplex für den Optimierungsprozess. Besonders die hohe Dimensionalität der Systemfunktion und somit der Zustände $x(t)$, die bei den üblichen kommerziellen Simulationsprogrammen aufgrund der Modellfeinheit und der eindimensionalen Diskretisierung der Strömungswege gegeben ist, machen die Lösung des Optimalsteuerungsproblems nach den Gleichungen (2.1a) bis (2.1c) schwer oder gar unmöglich. Eine andere Schwierigkeit ist in diesem Fall die fehlende Differenzierbarkeit einiger Teilmodelle im gesamten Systemmodell f. Ein Beispiel sind hier die intern benutzten Korrelationen für Wärmeübergangskoeffizienten, welche beim Umschlag von laminarer in turbulente Strömung je nach Implementierung nicht differenzierbar oder gar unstetig sind. Ein großes Augenmerk des zu konzeptionierenden Verfahrens sollte also auf die Bildung der zur Optimierung genutzten Systemfunktion f gerichtet werden. Dies sollte derart geschehen, dass die wesentlichen Eigenschaften des Prozesses, also der realen Systemfunktion f, noch genau genug beschrieben werden, jedoch Modellfeinheiten reduziert werden. In diesem Zusammenhang spricht man von der Bildung eines *Ersatzmodells* $\hat{f}$. Dieses Vorgehen bildet einen zentralen Bestandteil des im Rahmen dieser Arbeit entwickelten Verfahrens.

Zur Bildung eines solchen Ersatzmodells gibt es grundsätzlich eine Vielzahl an Möglichkeiten. So mag es Fälle geben, in denen die Prozesse simpel und durchschaubar genug sind, um a priori allein aus physikalischer Vorkenntnis ein einfaches, parameterbasiertes Ersatzmodell zu postulieren. Dessen inhärente Parameter wären dann mit den üblichen Methoden der Parameteridentifikation zu bestimmen. Dies ist bei größeren Systemen jedoch kaum machbar, da diese einerseits zu komplex sind, um alle Wechselwirkungen in einem Ersatzmodell zu erfassen und andererseits eine intuitive physikalische Argumentation allein oftmals nicht mehr ausreicht, um ein qualitativ gutes Modell zu erhalten. In einer derartigen Situation wäre es wünschenswert, auf eine allgemein anwendbare Struktur eines mathematischen Ersatzmodells $\hat{f}$ zurückgreifen zu können, die flexibel auf unterschiedlichste Systeme anwendbar ist.

Dies ermöglichen die sogenannten neuronalen Netzwerke, die letztlich eine gewichtete Superposition von nichtlinearen Basisfunktionen sind. Die entsprechenden Gewichte können im Rahmen einer Parameteridentifikation ermittelt werden. Neuronale Netzwerke sind in der Literatur äußerst prominent. Sie wurden bereits von vielen Forschungsgruppen aus unterschiedlichsten Fachrichtungen erfolgreich erprobt und sind für ihre Flexibilität und Lernfähigkeit bekannt (siehe zum Beispiel Diaconescu 2008, Kim u. a. 2004). Auch zur Black-Box-Systemidentifikation dynamischer Systeme sind sie gut geeignet, weswegen sie ein probates Mittel zur Bildung eines verlässlichen Ersatzmodells darstellen (Horvath 2003). Im Abschnitt 3.1 wird noch ausführlich auf verschiedene Arten von neuronalen Netzwerken und deren Theorie eingegangen. An dieser Stelle sei lediglich vorausgeschickt, dass derartige Ersatzmodelle interessanterweise eine beliebig genaue Approximationsfähigkeit besitzen, solange sie nur komplex genug gewählt sind. Darüber hinaus haben neuronale Netzwerke den Vorteil, dass sehr feine Details des Originalsystems durch die Black-Box-Modellierung ersetzt werden, wodurch im Allgemeinen eine massive Reduktion der Modellkomplexität erreicht wird. Gleichzeitig werden kritische Teile des Modells, wie etwa solche mit fehlender Differenzierbarkeit, geglättet.

Ist nun ein Ersatzmodell $\hat{f}$ gefunden, so kann die Lösung des Optimalsteuerungsproblems nach den Gleichungen (2.1a) bis (2.1c) angegangen werden. Hierzu sollen, wie bereits in Abschnitt 2.1 angedeutet, bereits vorhandene Implementierungen von Optimierungsverfahren herangezogen werden, um Zuverlässigkeit, Effizienz und Robustheit zu garantieren.

Um ein Verfahren entsprechend des vorstehend diskutierten Ansatzes zu formulieren, sollen im nun folgenden Kapitel zunächst notwendige theoretische Grundlagen über neuronale Netze als Ersatzmodelle

sowie die Eigenschaften und die Lösung von Optimalsteuerungsproblemen beleuchtet werden. Im Kapitel 4 werden die Überlegungen dieses und des nächsten Kapitels zur detaillierten Entwicklung des hier skizzierten Verfahrens aufgegriffen.

3 Theoretische Grundlagen

3.1 Neuronale Netzwerke als universales Ersatzmodell

Bei der Identifikation von komplexen Systemen mit vielen Variablen und Parametern spielt die Black-Box-Modellierung eine wichtige Rolle. Statt des Postulats eines physikalisch motivierten, parameterbehafteten Modells und der anschließenden Schätzung dieser Parameter derart, dass das Modell ein reales System hinreichend gut wiedergibt, wird das Modell bei der Black-Box-Vorgehensweise allein aus der Beobachtung des realen Systems – ohne vorheriges Wissen über die Modellstruktur – aufgebaut. Dies erfolgt im Allgemeinen durch eine gewichtete Summe von Basisfunktionen. Für die Basisfunktionen gibt es viele verschiedene Möglichkeiten. Eine dieser Möglichkeiten sind die sogenannten neuronalen Netze, die hierfür nichtlineare und sogenannte sigmoide Funktionen nutzen, welche weiter unten genauer erläutert werden (Ali u. Schmid 2010).

Neuronale Netze sind ein System von einer gewissen Anzahl von gleichen oder ähnlichen informationsverarbeitenden Einheiten, *Neuronen* genannt. Diese sind im Allgemeinen mit verschiedenen Gewichtungen untereinander verbunden und in einer gewissen Netzwerktopologie geordnet. Neuronale Netze zeichnen sich durch ihre Adaptivität aus: In einem iterativen Lernprozess können exogen vorgegebene Informationen erlernt werden, was sich in einer Adaption der netzinternen Gewichte ausdrückt (Horvath 2003).

Eine wichtige, nichtlineare Variante eines Neurons ist das sogenannte *Perzeptron*, welches Informationen sequentiell wie folgt verarbeitet. Zunächst werden die Eingangsinformationen $x \in \mathbb{R}^{n_x}$ gemäß $w^T x$ mit Gewichten $w \in \mathbb{R}^{n_x}$ versehen und linear kombiniert. In diese Linearkombination wird auch eine additive Konstante (engl. *bias*) einbezogen. Anschließend wird das Ergebnis dieser Superposition einer sogenannten Aktivierungsfunktion $\sigma(\,\cdot\,)$ übergeben, die die Ausgangsinformation des Perzeptrons erzeugt. Insgesamt ergibt sich also für ein Perzeptron die Modellgleichung:

$$y = \sigma\left(w^{\mathrm{T}} x\right) \tag{3.1}$$

Die Aktivierungsfunktion σ ist hierbei zumeist eine nichtlineare, monoton steigende, differenzierbare und beschränkte Funktion. Diese Gruppe von Funktionen wird sigmoide Funktionen genannt (Horvath 2003). Die wichtigsten Vertreter sind die logistische Funktion,

$$\operatorname{logsig}(x) := \frac{1}{1+\mathrm{e}^{-x}}\,, \tag{3.2}$$

sowie die hyperbolische Tangensfunktion,

$$\tanh(x) := \frac{1-\mathrm{e}^{-2x}}{1+\mathrm{e}^{-2x}}\,. \tag{3.3}$$

Alternativ können auch sogenannte radiale Basisfunktionen (etwa die Gaussche Glockenkurve) als Aktivierungsfunktion dienen. Perzeptronen nach Gleichung (3.1) sind die Grundbausteine für unterschiedlichste Klassen von neuronalen Netzwerken (Haykin 1999).

Eine oft verwendete Architektur von Neuronalen Netzen sind die sogenannten Mehrschicht-Perzeptronen (engl. *multilayer perceptrons – MLP*). Dabei handelt es sich um ein vorwärtsgerichtetes, nicht-zurückführendes (*feed-forward*) Netzwerk, das aus einer Eingangsschicht (engl. *input layer*), mindestens einer versteckten Schicht (engl. *hidden layer*) und einer Ausgangsschicht (engl. *output layer*) besteht.

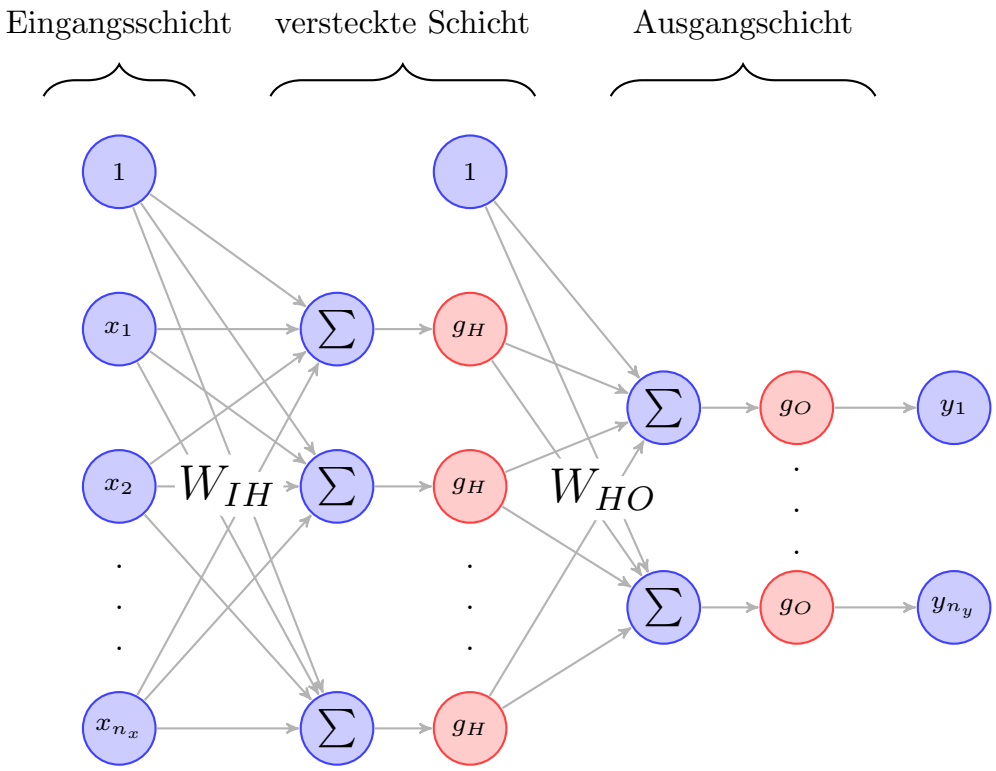


Abbildung 3.1.: Aufbau eines Mehrschicht-Perzeptrons mit Aktivierungsfunktionen g_H in der versteckten beziehungsweise g_O in der Ausgangschicht.

Dabei sind alle Neuronen jeder Schicht mit allen Neuronen der direkt benachbarten Schichten verbunden (Horvath 2003). Die Grundstruktur dieser Netzwerke ist (für ein Netz mit nur einer versteckten Schicht) in Abbildung 3.1 dargestellt.

Im Allgemeinen sind die Aktivierungsfunktionen in der versteckten und der Ausgangsschicht, $g_O(\cdot)$ beziehungsweise $g_H(\cdot)$, sigmoide Funktionen. Je nach Modellierungsaufgabe oder Genauigkeitsansprüchen kommen hier teilweise aber auch lineare Funktionen in Frage. Entsprechend der Netzwerktopologie und der Modellgleichungen für ein MLP ergibt sich (im Falle einer versteckten Schicht) für die k-te Ausgangsgröße eines MLPs (Haykin 1999, Horvath 2003):

$$y_k = g_O\left(\sum_{i=0}^{n_y} w_{HO,k,j} \cdot g_H\left(\sum_{i=0}^{n_x} w_{IH,j,i} \cdot x_i\right)\right) \tag{3.4}$$

Mehrschicht-Perzeptronen dieser Form haben eine insbesondere für die nichtlineare Systemidentifikation sehr attraktive Eigenschaft: Theoretische Untersuchungen haben gezeigt, dass ein MLP gemäß Abbildung 3.1 und Gleichung (3.4) mit einer ausreichenden Anzahl von Neuronen in der versteckten Schicht in der Lage ist, jede stetige Funktion $f : \mathbb{R}^n \to \mathbb{R}$ mit beliebig hoher Genauigkeit zu approximieren. Es konnte darüber hinaus gezeigt werden, dass hierfür nur eine versteckte Schicht mit nichtlinearen Neuronen (also mit nichtlinearen Aktivierungsfunktionen σ) sowie ein einfacher linearer, gewichtender Addierer in der Ausgangsschicht ausreichend sind. Aufgrund dieser Eigenschaft werden neuronale Netze, die auf Mehrschicht-Perzeptronen basieren, als universaler Approximator bezeichnet (Cybenko 1989, Doya 1993, Schäfer u. Zimmermann 2006).

3.1.1 Zeitdiskrete NARX-Netzwerke

Statische neuronale Netzwerke, wie die im vorigen Abschnitt eingeführten Mehrschicht-Perzeptronen, reichen in der Praxis meistens nicht aus, um das Verhalten von dynamischen Systemen hinreichend zu

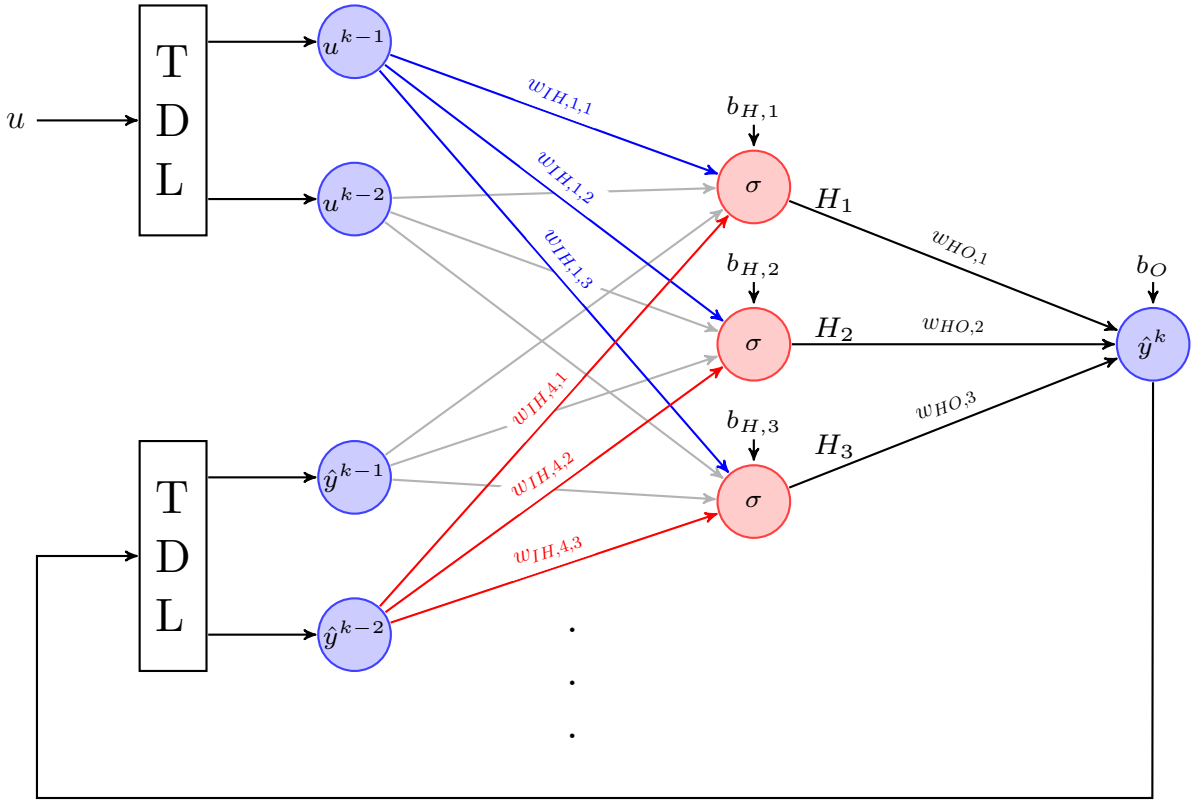


Abbildung 3.2.: Topologie von zeitdiskreten NARX-Netzwerken mit einer versteckten Schicht.

beschreiben. Daher erfolgt in diesem Abschnitt eine Erweiterung dieser Netze auf Netzwerktopologien mit dynamischen Eigenschaften.

Hierzu wird einerseits ein Zwischenspeicher in die Netze eingefügt. Diese werden *tapped delay lines (TDL)* genannt und ermöglichen es, statt den Eingangsgrößen des aktuellen Zeitschritts auch Werte der letzten d_u beziehungsweise d_y Zeitschritte dem neuronalen Netzwerk zuzuführen und somit in die Berechnung der Ausgangsgröße mit einzubeziehen. Darüber hinaus führt man, abhängig von der gewählten Topologie des dynamischen Netzes, eine Rückführung (engl. *feed-back*) der Systemantwort in den Systemeingang ein. Erst durch diesen Schritt bekommt das neuronale Netz die Möglichkeit, Systeme mit einer gewissen Eigendynamik abzubilden (Horvath 2003).

Werden diese Erweiterungen in ein Mehrschicht-Perzeptron implementiert, so erhält man eine spezielle Topologie von Netzwerken, welche *NARX*-Netzwerke *(nonlinear autoregressive exogenous)* genannt werden. Entsprechend der vorstehenden Ausführungen ergibt sich die allgemeine Definitionsgleichung von NARX-Netzwerken zu (Horvath 2003):

$$\hat{y}^k = f\left(u^{k-1}, u^{k-2}, \ldots, u^{k-d_u}, \hat{y}^{k-1}, \hat{y}^{k-2}, \ldots, \hat{y}^{k-d_y}\right) \tag{3.5}$$

Abbildung 3.2 veranschaulicht die Architektur dieser Art von neuronalen Netzwerken. Anhand dieser Illustrierung kann der interne Berechnungsvorgang im neuronalen Netz wie folgt beschrieben werden. Die Eingabegröße u wird über die tapped delay line in das eigentliche neuronale Netz eingespeist. Dies hat zur Folge, dass auch u-Werte aus der Vergangenheit in die Ermittlung der aktuellen Systemantwort einbezogen werden können. Wie viele Schritte in die Vergangenheit der Speicher reicht, wird gemäß Gleichung (3.5) durch den Parameter d_u bestimmt. Analog wird der Systemoutput $\hat{y}$ der letzten d_y Zeitschritte zur Bestimmung der aktuellen Ausgangsgröße in den Eingang zurückgeführt, was charakteristisch für die NARX-Topologie ist. Aus den tapped delay lines gelangen die Eingabewerte in die Eingangsschicht des Netzwerks, von wo sie, mit einem Gewicht $w_{IH,j,i}$ skaliert, in die versteckte Schicht übertragen werden. Dort werden die gewichteten Inputs sowie eine additive Konstante $b_{H,i}$ der Aktivierungsfunktion σ übergeben. Die Ausgaben der versteckten Schicht werden dann ihrerseits wieder mit einem Gewicht $w_{HO,i}$

versehen und zusammen mit einer weiteren additiven Konstante b_O linear kombiniert, um schlussendlich den Output $\hat{y}^k$ des neuronalen Netzes zum aktuellen, k-ten Zeitschritts zu bilden.

Entsprechend dieser internen Berechnungsstruktur ergibt sich folgende Bestimmungsgleichung für die Ausgangsgröße eines NARX-Netzwerks:

$$\hat{y}^k = \sum_{i=1}^{N} w_{\mathrm{HO},i} \cdot \underbrace{\tanh\left[\sum_{j=1}^{N} w_{\mathrm{IH},j,i} \cdot \mathrm{inp}_j + b_{\mathrm{H},i}\right]}_{=:H_i} + b_{\mathrm{O}} \tag{3.6}$$

Hierbei bezeichnet inp_j die j-te Eingangsgröße in das Netz in der allgemeinen Form von Gleichung (3.5). Mit H_i wird abkürzend der Rückgabewert des i-ten Neurons in der versteckten Schicht bezeichnet. Gleichung (3.6) gilt für ein Netz mit einer Ausgangsgröße, lässt sich aber analog für mehrere Ausgangsgrößen formulieren.

Die korrekte Bestimmung der Netzparameter w_{HO}, w_{IH} und b_O und b_H ist Aufgabe des Trainingsverfahrens, welches im Abschnitt 3.1.3 erläutert wird. Dabei ist zu beachten, dass die Anzahl an Parametern n_η im neuronalen Netz bei n_h Neuronen in der versteckten Schicht und d_u beziehungsweise d_y gespeicherten zurückliegenden Zeitschritten in den tapped delay lines durch $n_\eta = 1 + n_h \cdot (2 + d_u + d_y)$ bestimmt ist. Je komplexer das Netzwerk ist, desto mehr adaptierbare Parameter hat es inne und desto komplexer ist die abbildbare Funktion. In diesem Sinne gilt, wie auch bei statischen Netzen, für dynamische, zurückführende neuronale Netze das Theorem des universalen Approximators, das besagt, dass mit einer angemessenen Anzahl von Neuronen in der versteckten Schicht jedes dynamische System beliebig genau approximieren werden kann (Schäfer u. Zimmermann 2006, Doya 1993). Diese Aussage ist die Erweiterung des Approximatortheorems für statische Netze auf deren dynamische Pendants.

Neben der NARX-Topologie gibt es noch einige andere Architekturen für dynamische neuronale Netzwerke in diskreter Zeit. Diese unterscheiden sich im Wesentlichen dadurch, ob sie die letzten d_u Zeitschritte der Eingangsgröße u oder zusätzlich die zurückgeführten Ausgangsgrößen speichern und erneut in das neuronale Netz einspeisen (Horvath 2003). Da für die Anwendung in dieser Arbeit ein möglichst allgemeiner Fall abgedeckt werden soll, wird unter den zeitdiskreten Netztopologien, wie sie in diesem Kapitel diskutiert wurden, die NARX-Topologie bevorzugt, weil sie sowohl Eingangs- und Ausgangsgrößen dynamisch mit berücksichtigt und somit prinzipiell eine möglichst gute Näherung an die Realität darstellt.

Während in den bisherigen Abhandlungen ausschließlich zeitdiskrete neuronale Netze diskutiert wurden, soll im nächsten der Fokus auf zeitkontinuierliche Netze gelegt werden.

3.1.2 Zeitkontinuierliche CTRNN-Netze

Alle bisher behandelten neuronalen Netze sind diskret in der Zeit. Je nach Verwendungszweck kann es jedoch vorkommen, dass man ein zeitstetiges neuronales Netz benötigt oder bevorzugt. Für diese Aufgabe gibt es sogenannte zeitstetige, rückführende neuronale Netzwerke (engl. *continuous time recurrent neural networks*, kurz CTRNN). Insbesondere bei der Systemidentifikation von Prozessen, welche naturbedingt kontinuierlichen Charakter haben, können CTRNNs gegenüber diskreten Varianten wie etwa NARX-Netzen vorteilhaft sein.

Ein besonders großer Vorteil von CTRNN liegt darin, dass sie im Gegensatz zur diskreten NARX-Topologie keine implizite Sampling-Zeit $\triangle t$ innehaben. Das bedeutet insbesondere, dass sie nicht neu trainiert werden müssen, wenn die Sampling-Zeit gewechselt werden soll, etwa zur Anpassung der zeitlichen Auflösung der Trainingsdaten oder der Vorhersagen. Darüber hinaus können sie einen Genauigkeitsvorteil bieten, wenn die Weiterverwendung ein zeitstetiges Ersatzmodell erfordert, da ein CTRNN dieses direkt bereitstellen kann, bei einem NARX-Netzwerk jedoch Zwischenschritte nötig sind[1]. Allerdings ist

[1] Welche Zwischenschritte dies sind und wann sie nötig werden, wird in Kapitel 4 detailliert erläutert.

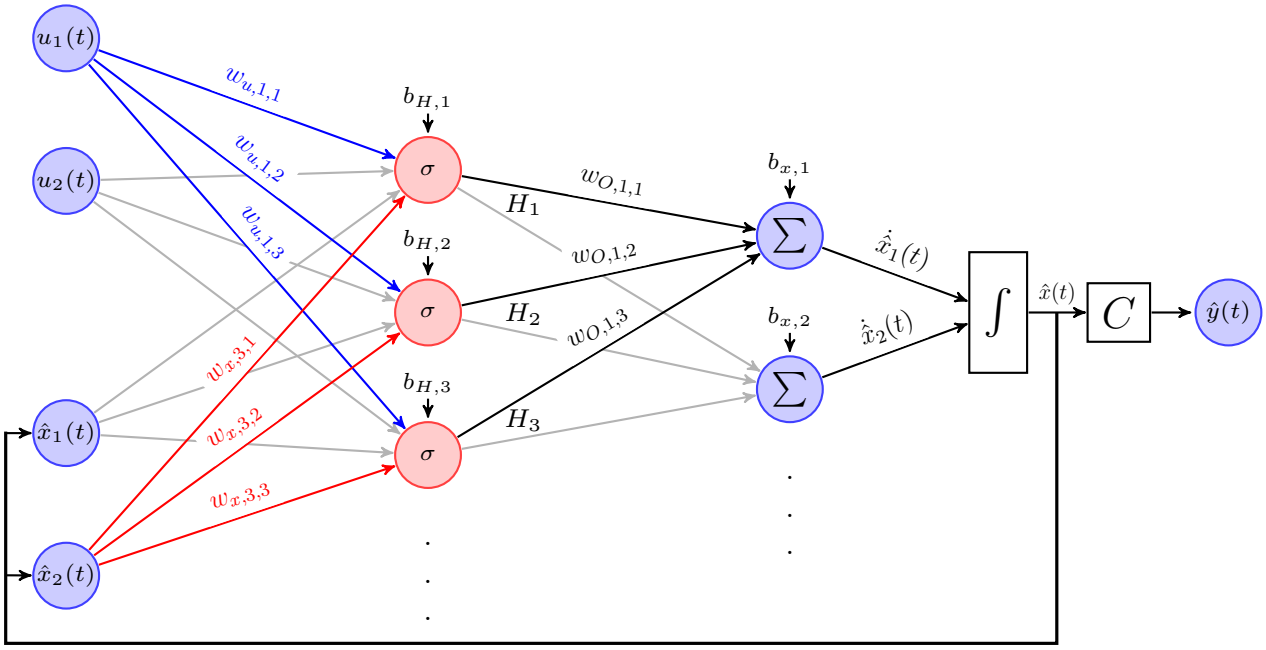


Abbildung 3.3.: Topologie eines zeitkontinuierlichen, rückführenden neuronalen Netzwerks nach den Gleichungen (3.7a) und (3.7b).

zu erwarten, dass ein CTRNN im Allgemeinen ein mathematisch komplexeres Ersatzmodell bilden wird, da es, wie weiter unten ausführlich erläutert, aus gekoppelten Differentialgleichungen besteht.

In der Literatur sind im Wesentlichen zwei verschiedene Formen von CTRNNs präsent. Die erste Form ist der Bestimmungsgleichung für zeitdiskrete NARX-Netze, Gleichung (3.6), auffallend ähnlich und lautet (Al Seyab 2006, Cao u. Al Seyab 2006):

$$\dot{\hat{x}}(t) = W_O \cdot \sigma\left(W_x \hat{x}(t) + W_u u(t) + b_H\right) + b_O \tag{3.7a}$$

$$\hat{y}(t) = C \cdot \hat{x}(t) \tag{3.7b}$$

Die Gleichungen (3.7a) und (3.7b) bilden ein zeitkontinuierliches System und liegen in Zustandsraumdarstellung vor. Die Struktur des Netzes ist in Abbildung 3.3 dargestellt.

Für die folgenden Betrachtungen bestehe das CTRNN aus n_u Eingängen, n_h Neuronen in der versteckten Schicht, n_x Zuständen und n_y Ausgängen. Dann sind $u(t) \in \mathbb{R}^{n_u}$ der Steuerungsvektor, $\hat{y}(t) \in \mathbb{R}^{n_y}$ die Ausgangsgrößen und $\hat{x}(t) \in \mathbb{R}^{n_x}$ die internen Zustände des Netzes. Die Parameter des Netzes sind durch die Gewichte $W_x \in \mathbb{R}^{n_h \times n_x}$, $W_u \in \mathbb{R}^{n_h \times n_u}$ und $W_O \in \mathbb{R}^{n_x \times n_h}$ sowie die additiven Konstanten $b_H \in \mathbb{R}^{n_h}$ und $b_O \in \mathbb{R}^{n_x}$ gegeben. Zusammen bilden sie die Gesamtheit der freien Parameter des Netzes und können nach

$$\eta = \left[\mathrm{vec}\left(W_u\right)^T \;\; \mathrm{vec}\left(W_x\right)^T \;\; \mathrm{vec}\left(W_O\right)^T \;\; b_H^T \;\; b_O^T\right]^T \tag{3.8}$$

zum Parametervektor $\eta \in \mathbb{R}^{n_\eta}$ zusammengefasst werden (Al Seyab 2006). Dieser umfasst dann $n_\eta = n_x\left(n_h + 1\right) + n_h\left(n_x + n_u + 1\right)$ Parameter. Zur Ermittlung der Ausgangsgrößen $\hat{y}(t)$ aus den internen Zuständen $\hat{x}(t)$ dient die Matrix

$$C = \left[I_{n_y \times n_y}, 0_{n_y \times \left(n_x - n_y\right)}\right], \tag{3.9}$$

aus der sich ergibt, dass die Ausgangsgrößen gerade die ersten n_y internen Zustände des Netzes sind (Al Seyab 2006). Als Aktivierungsfunktion σ in zeitstetigen neuronalen Netzen wird im Rahmen dieser Arbeit wie auch im zeitdiskreten Fall der Tangens Hyperbolicus genutzt, was in der Literatur auch üblich ist (Beer 1995, Chen u. Wermter 1998, Chow u. Li 2000, Li u. a. 2005).

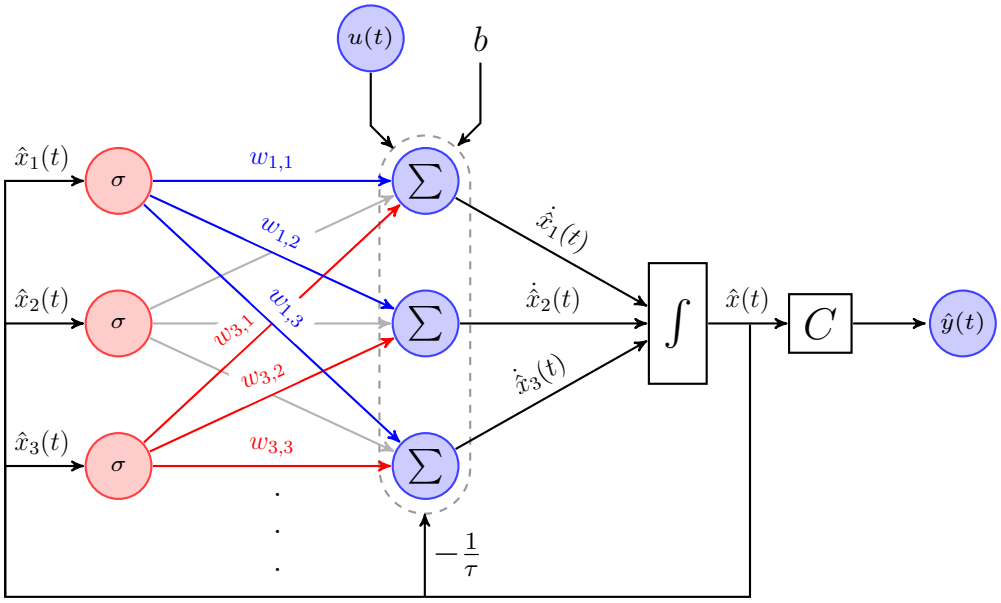


Abbildung 3.4.: Topologie eines zeitkontinuierlichen, rückführenden neuronalen Netzwerks nach den Gleichungen (3.10a) und (3.10b).

CTRNN-Netze gemäß den Gleichungen (3.7a) und (3.7b) wurden bereits erfolgreich dazu benutzt, um Black-Box-Modelle eines Zwangumlaufverdampfers und zweier gekoppelter chemischer Reaktoren zu bilden um sie anschließend im Rahmen einer modellprädiktiven Regelung weiterzuverwenden. Hierbei hat eine Netzkomplexität von $n_x = 5, n_h = 8$ beziehungsweise $n_x = 6, n_h = 6$ bei einer Kreuzvalidierung sehr gute Ergebnisse geliefert (Al Seyab u. Cao 2008, Al Seyab 2006).

Eine zweite prominente Form von CTRNN-Netzen, die in der Literatur häufig anzutreffen ist, basiert wie die vorgenannte Form auf der Zustandsraumdarstellung eines dynamischen Systems und kann wie folgt formuliert werden (Potter 2006):

$$\dot{\hat{x}}(t) = -T \cdot \hat{x}(t) + W \cdot \sigma(\hat{x}(t)) + B + I(t) \tag{3.10a}$$

$$\hat{y}(t) = C \cdot \hat{x}(t) \tag{3.10b}$$

In den Gleichungen (3.10a) und (3.10b) ist $W \in \mathbb{R}^{n_x \times n_x}$ die Matrix der Gewichte des neuronalen Netzes, $B \in \mathbb{R}^{n_x}$ der Vektor der additiven Konstanten und $T \in \mathbb{R}^{n_x \times n_x}$ eine Diagonalmatrix nach

$$T = \operatorname{diag}\left\{\tau_1^{-1}, \tau_2^{-1}, \ldots\right\}, \tag{3.11}$$

wobei τ_i eine charakteristische Zeitkonstante des i-ten Neurons ist. Diese bildet, gemeinsam mit W und B, die freien Parameter $\eta \in \mathbb{R}^{n_\eta}$ des Netzes mit $n_\eta = n_x^2 + 2 \cdot n_x$ (Potter 2006). Die grundsätzliche Struktur der Netztopologie ist in Abbildung (3.4) dargestellt.

C ist eine Matrix, die den Ausgängen $\hat{y}(t)$ des neuronalen Netzes die jeweiligen interne Zustände $\hat{x}(t)$ zuordnet und wird, ebenso wie bei der erstgenannten Form von CTRNNs, nach Gleichung (3.9) gesetzt. Der Vektor $I(t)$ beinhaltet die Eingangsgröße in jedes Neurons des Netzes. Hierbei ist es sinnvoll, die Steuerungen $u(t)$ einem (oder mehreren) der letzten Elemente von $I(t)$ zuzuordnen, da die ersten Elemente der vektoriellen Gleichung (3.10a) aufgrund der Definition der Matrix C nach Gleichung (3.9) die Ausgangsgrößen sind. Durch das Zuordnen der Steuerung $u(t)$ zum letzten Element der Vektorgleichung ist eine möglichst starke Verkopplung der Steuerung mit der Systemdynamik sichergestellt.

CTRNNs wurden in der Literatur bisher wenig zur Systemidentifikation eingesetzt, sondern eher auf theoretischer Ebene untersucht. Wie für die zeitdiskreten Varianten konnte auch für CTRNN-Netze

theoretisch nachgewiesen werden, dass jedes beliebige dynamische System der Form $\dot{x} = f(x,u,t)$ durch ein CTRNN der Form (3.10a) und (3.10b) beliebig genau approximiert werden kann, wenn nur ausreichend Neuronen im Netz vorgesehen werden (Nakamura u. Nakagawa 2009). Insofern besitzen derartige CTRNN-Netze die wünschenswerte Eigenschaft eines universalen Approximators. Während diese Aussage lange nur für ungesteuerte, autonome Systeme der Form $\dot{x} = f(x)$ theoretisch bewiesen werden konnte (Funahashi u. Nakamura 1993), wurde sie im Laufe der Jahre nach und nach erweitert. Relativ früh gelang die Erweiterung der Beweisführung auf gesteuerte, allerdings nur eingangsaffine Systeme der Form $\dot{x} = f(x) + g \cdot u$. (Delgado u. a. 1995). Zur Jahrtausendwende konnten die Beweise dann auf allgemeinere eingangsaffine Systeme der Form $\dot{x} = f(x) + g(x) \cdot u$ erweitert werden (Kambhampati u. a. 2000). Erst einige Jahre später gelang es, den theoretischen Beweis für allgemeine, nicht-autonome dynamische Systeme der Form $\dot{x} = f(x,u,t)$ zu liefern (Chow u. Li 2000, Li u. a. 2005, Nakamura u. Nakagawa 2009).

Für zeitstetige neuronale Netze nach den Gleichungen (3.10a) und (3.10b) sind derartige theoretische Beweise bisher noch nicht geführt worden. Dennoch ist aus den bereits erwähnten Ergebnissen ersichtlich, dass diese Form von CTRNNs physikalische Systeme sehr gut modellieren können (Al Seyab 2006).

3.1.3 Trainingsverfahren für neuronale Netzwerke

Um einem neuronalen Netz Wissen oder Intelligenz aufzuprägen, muss es trainiert werden. Trainieren bedeutet das Adaptieren von Netzwerkgewichten, sodass das Netz das gewünschte reale System angemessen nachbilden kann. Vor dem Training muss die Topologie des Netzes, also insbesondere die Anzahl der Neuronen in der versteckten Schicht festgelegt sein. Im Falle von zeitdiskreten NARX-Netzen sind auch die Delay-Werte d_u und d_y vorher zu wählen. Dadurch ist die Anzahl der freien Parameter im Netz bestimmt, die im Zuge des Trainings möglichst gut adaptiert werden sollen.

Beim Training eines neuronalen Netzwerks handelt es sich um ein Ausgleichsproblem. Es werden genau jene Parameter η gesucht, die ein Gütekriterium minimieren, welches die Abweichung der Vorhersage des Netzes von den Werten des realen Systems bewertet. In diesem Sinne ist ein Trainingsdatensatz für das Trainieren von neuronalen Netzen zwingend erforderlich. Als Gütekriterium dient für gewöhnlich die Summe der quadratischen Abweichungen zwischen der Vorhersage des Netzes $\hat{y}$ und dem Trainingswert y an jedem der N Datenpunkte im Trainingssatz (Horvath 2003):

$$\varphi(\eta) = \frac{1}{2}\sum_{i=1}^{N} \epsilon^2(\eta) = \frac{1}{2}\sum_{i=1}^{N} \left(y_i - \hat{y}_i(\eta)\right)^2 \tag{3.12}$$

Das Gütekriterium nach Gleichung (3.12) ist eine stetige Funktion des Parametervektors. Somit beschreibt sie eine multidimensionale Fehleroberfläche im Parameterraum, deren Minimum im Zuge des Trainingsprozesses gesucht wird. Bei neuronalen Netzen, die auf nichtlinearen Aktivierungsfunktionen basieren, besitzt die Fehleroberfläche jedoch eine große Menge von lokalen Minima, weil die Ausgangsgröße des Netzes und somit auch das Gütekriterium nichtlinear von den Parametern abhängt und darüber hinaus noch einmal im Gütekriterium quadriert wird, was eine hohe Nichtlinearität des Gütekriteriums bezüglich der Parameter bewirkt. Die Herausforderung im Trainingsprozess ist es nun, unter diesen lokalen Minima möglichst das globale Minimum zu finden, welches der bestmöglichen Wahl der Netz-Parameter entspricht.

Hierfür ist eine Fülle von ganz unterschiedlichen Verfahren entstanden, die im Folgenden näher beleuchtet werden sollen. Dabei wird zwischen Lernalgorithmen für zeitdiskrete NARX-Netzwerke und für zeitstetige CTRNN-Netze unterschieden.

Lernverfahren für neuronale Netze in diskreter Zeit

Wie in der Optimierung üblich, werden zur Minimierung der Gütefunktion (3.12) iterative, meist gradientenbasierte Verfahren eingesetzt, da diese als robust gelten und gegenüber gradientenfreien, metaheuristischen Suchverfahren über wesentlich bessere Konvergenzeigenschaften verfügen (Papageorgiou

2006). Dem steht allerdings der Nachteil entgegen, dass Gradientenverfahren beim iterativen Minimierungsprozess in einem lokalen Minimum hängen bleiben können. Somit garantieren diese Verfahren keine globale Optimalität, weswegen es sich anbietet, das Training mehrfach mit unterschiedlichen Anfangswerten zu wiederholen, wovon man sich erhofft, dass unter den gefundenen Minima das globale Minimum enthalten ist. Dementsprechend läuft der Trainingsprozess von dynamischen neuronalen Netzen so ab, dass für die aktuelle beste Parameterschätzung die zeitabhängigen Ausgangsgrößen eines Netzes bei gegebenen Eingangsgrößen bestimmt werden, diese anschließend mit dem Verhalten des realen Prozesses verglichen werden und daraufhin in einer verfahrensabhängigen Art und Weise eine neue, bessere Parameterschätzung erfolgt, wodurch eine neue Iteration beginnt.

Zu beachten ist, dass im Fall von NARX-Netzwerken der mit der Rückkopplung hergestellte Kreis (vergleiche Abbildung 3.2) während des Trainings noch nicht geschlossen ist. Das bedeutet, dass zu dieser Zeit das NARX-Netzwerk formal ein einfaches vorwärtsgerichtetes Netzwerk ist, was das Training erheblich erleichtert. Echte rekurrente Netze mit Feedback haben beim Training den gravierenden Nachteil, dass sich aufgrund der Rückkopplung Instabilitäten entwickeln können, sodass das Training besonderer Verfahren bedarf (Horvath 2003).

Ein berühmtes und sehr wichtiges Verfahren zum Lösen derartiger nichtlinearer Ausgleichsprobleme mit quadratischen Gütekriterien ist das Levenberg-Marquardt-Verfahren, welches bereits 1963 formuliert wurde (Marquardt 1963). Diese Methode konnte mit großem Erfolg auf den Lernprozess von neuronalen Netzwerken übertragen werden (Hagan u. Menhaj 1994). Aufgrund seiner Flexibilität und Robustheit soll es auch im Rahmen dieser Arbeit zum Training von NARX-Netzen eingesetzt werden.

Die Grundidee des Levenberg-Marquardt-Verfahrens liegt in der Kombination eines Quasi-Newton-Verfahrens mit einem Gradientenabstiegsverfahren zur Berechnung der besten Ausgleichslösung für die Netz-Parameter η (Papageorgiou 2006). Dabei macht es sich die Tatsache zunutze, dass bei Gütekriterien in der Form von summierten quadratischen Abweichungen der Gradient $\nabla\varphi(\eta)$ sowie die Hessematrix H_φ der Gütefunktion mit der Jacobi-Matrix der Residuen

$$J_\epsilon(\eta) := \frac{\partial \epsilon(\eta)}{\partial \eta} \tag{3.13}$$

wie folgt formuliert werden können (Papageorgiou 2006):

$$\nabla\varphi(\eta) = J_\epsilon^T(\eta)\,\epsilon \tag{3.14}$$

$$H_\varphi(\eta) \approx J_\epsilon^T(\eta)\,J_\epsilon(\eta) \tag{3.15}$$

Die Gleichungen (3.14) und (3.15) machen deutlich, dass durch die Auffassung des Trainingsproblems als eine Minimierung von Abweichungsquadraten ein rechenzeiteffizientes Verfahren konstruiert werden kann, da alleine aus Ableitungsinformationen erster Ordnung auch jene zweiter Ordnung gewonnen werden können, ohne dass die Hesse-Matrix explizit berechnet werden muss. Die Jacobi-Matrix nach Gleichung (3.13), die die Sensitivität des Modellfehlers bezüglich der verschiedenen Parameter η beschreibt, lässt sich hierbei durch ein Verfahren, das als *Rückpropagierung* (engl. *backpropagation*) des Fehlervektors ϵ in das neuronale Netz bekannt ist, aus dem Netz berechnen. Dieses beruht auf der Differenziation des Vorhersagefehlers nach den Parametern η mithilfe der Kettenregel und der anschließenden Auswertung dieser Beziehung anhand der aktuellen Gewichte im neuronalen Netz (Haykin 1999).

Konstruiert man mit dem Gradienten und der Hessematrix nach Gleichung (3.14) beziehungsweise (3.15) ein Verfahren, das eine mit dem sogenannten Dämpfungsfaktor μ gewichtete Kombination aus einem Quasi-Newton-Verfahren und einem Gradientenabstiegsverfahren darstellt, erhält man die Iterationsvorschrift des Trainingsprozesses zu (Al Seyab 2006, Hagan u. Menhaj 1994)

$$\eta_{k+1} = \eta_k - \left[J_\epsilon^T(\eta)\,J_\epsilon(\eta) + \mu_k I\right]^{-1} \cdot J_\epsilon^T(\eta)\,\epsilon \quad . \tag{3.16}$$

Abhängig vom iterationsabhängigen Dämpfungsfaktor μ_k ändert der Iterationsprozess nach Gleichung (3.16) sein Verhalten. Einerseits liegt für sehr kleine μ_k ein reines Quasi-Newton-Verfahren mit approximierter Hesse-Matrix vor. Dieses hat die wünschenswerte Eigenschaft, in der Nähe des Minimums von φ eine quadratische Konvergenzgeschwindigkeit aufzuweisen, konvergiert jedoch nur für hinreichend gute Startwerte. Für große μ_k wird die Iterationsvorschrift dagegen ein Gradientenabstiegsverfahren, welches für seine Robustheit in Bezug auf Parameterschätzwerte weit von der optimalen Lösung bekannt ist. Durch eine geeignete, dynamische Anpassung des Dämpfungsfaktors μ_k kann somit die Konvergenz des Verfahrens gesteuert werden. Um zunächst iterativ in die Nähe des Minimums zu gelangen, wird das Verfahren zu Beginn mit einem großen μ_k durchlaufen. Im Laufe der nächsten Iterationen kann μ_k dann schrittweise verringert werden, um schnellstmöglich zum schnell konvergierenden Quasi-Newton-Verfahren überzugehen. Die absoluten Beträge eines „großen" oder „kleinen" Parameters μ_k sind jedoch problemabhängig und können anhand der Eigenwerte der Hessematrix $H_\varphi(\eta)$ oder im Einzelfall notfalls heuristisch ermittelt werden (Transtrum u. Sethna 2012).

Äußerst wichtig für die Konvergenz des Levenberg-Marquardt-Verfahrens ist die dynamische Anpassung des Dämpfungsfaktors während des Iterationsvorgangs. Das gängigste und einfachste Mittel ist hier, nach einem erfolgreichen Iterationsschritt, der das Gütekriterium φ tatsächlich verringert, μ_k um einen festen Faktor zu reduzieren. Analog dazu wird μ_k um den gleichen Faktor wieder vergrößert, wenn die Iteration nicht erfolgreich war (Moré 1978, Lampton 1997, Transtrum u. Sethna 2012). In diesem Fall wird dann natürlich auch die jeweilige Parameterschätzung η_k verworfen. Typische Werte für diese Faktoren werden in der Literatur zwischen 0.1 und 0.75 angegeben (Nielsen 1999, Transtrum u. Sethna 2012). Untersuchungen zeigten jedoch, dass dieses Vorgehen zu einer nichtmonotonen Verringerung von μ_k im Laufe der Iterationen und somit zu langsamerer Konvergenz führen kann. Um dies zu vermeiden, wurde glattere Anpassungsregeln für μ_k im Falle einer erfolgreichen Iteration vorgeschlagen (Nielsen 1999). Auf eine detaillierte Erläuterung dieser Anpassregeln soll an dieser Stelle verzichtet werden, da das im Rahmen dieser Arbeit genutzte Levenberg-Marquardt-Verfahren bereits in implementierter Form vorliegt und über eigene Anpassregeln verfügt. Dies wird in Kapitel 4 näher beleuchtet.

Lernverfahren für zeitstetige neuronale Netze

Zum Training von CTRNN, welche zeitkontinuierlich sind, kann prinzipiell ebenso wie bei NARX-Netzwerken das Levenberg-Marquardt-Verfahren zum Einsatz kommen. Dies ist jedoch weitaus aufwändiger als im zeitdiskreten Fall, da die Sensitivitäten nach Gleichung (3.13) wegen der differentiellen Form des Netzes nun nicht mehr durch eine Rückpropagierung des Fehlers durch das neuronale Netz ermittelt werden können.

Daher ist es nötig, die Sensitivitäten auf anderem Wege zu berechnen. Ein oft genutztes Verfahren hierzu ist eine Sensitivitätsanalyse, die während der im Training ohnehin nötigen Integration der CTRNN-Gleichungen[2] zusätzlich die sogenannten *Sensitivitätsgleichungen* löst, welche direkt aus der Differentiation der CTRNN-Gleichungen nach dem Parametervektor η folgen, wobei zu beachten ist, dass während des Trainingsprozesses $u(t)$ festgelegt ist (Cao 2005, Cao u. Al Seyab 2006):

$$\frac{\partial}{\partial t}\left(\frac{\partial \hat{x}\,(t)}{\partial \eta}\right) = \frac{\partial f}{\partial \hat{x}} \cdot \frac{\partial \hat{x}\,(t)}{\partial \eta} + \frac{\partial f}{\partial \eta} \tag{3.17a}$$

$$\frac{\partial \hat{y}\,(t)}{\partial \eta} = C \cdot \frac{\partial \hat{x}\,(t)}{\partial \eta} \tag{3.17b}$$

Hierin bezeichnet f die rechte Seite von Gleichung (3.7a) beziehungsweise von Gleichung (3.10a). Die für des Levenberg-Marquardt-Verfahren nötige Jacobimatrix lässt sich nun wie folgt zusammensetzen (Al Seyab 2006):

$$J_\epsilon(\eta) = \left[J_1^T(\eta) \ \ldots \ J_N^T(\eta)\right] \tag{3.18}$$

[2] Die folgenden Ausführungen sind sowohl für CTRNN-Netze nach den Gleichungen (3.7a) und (3.7b) als auch für die alternative Form der Gleichungen (3.10a) und (3.10b) gültig.

Hierin ist jeder Block J_i die Jacobimatrix der Residuen ϵ_i bezüglich der Parameter η zum i-ten Zeitschritt und hat die Dimension $n_y \times n_\eta$. Sie berechnet sich nach (Al Seyab 2006)

$$J_i(\eta) = \frac{\partial \epsilon_i}{\partial \eta} = \frac{\partial \hat{y}(t_i,\eta)}{\partial \eta} = C \cdot \frac{\partial \hat{x}(t_i,\eta)}{\partial \eta} \quad . \tag{3.19}$$

Bei den Gleichungen (3.17a) und (3.17b) handelt es sich um ein lineares, zeitvariantes System von Differentialgleichungen, das wie bereits erwähnt zusammen mit den CTRNN-Gleichungen numerisch integriert werden muss. Dies kann, abhängig von der Größe des CTRNNs, sehr rechenaufwändig werden, da die Anzahl der zu lösenden Differentialgleichungen von n_x auf $n_x \cdot (1+n_\eta)$ steigt (Al Seyab 2006). Eine Möglichkeit, diesen Prozess effizienter zu gestalten, ist eine Lösung der Sensitivitätsgleichungen auf Basis von Taylor-Reihen-Entwicklungen der auftretenden Variablen in Verbindung mit automatischer Differentiation, welche jedoch einen recht hohen Implementierungsaufwand erfordert (Atuonwu u. a. 2010, Cao 2005, Cao u. Al Seyab 2006). Wie sich in Untersuchungen herausgestellt hat, ist dieses Vorgehen wesentlich rechenzeiteffizienter und darüber hinaus auch genauer als die Lösung der traditionellen Sensitivitätsgleichungen (3.17a) und (3.17b) (Al Seyab 2006, Al Seyab u. Cao 2008).

Um das grundsätzliche Problem der rechenintensiven Berechnung der dynamsichen Sensitivitäten nach den Gleichungen (3.17a) und (3.17b) zu umgehen, können sogenannte metaheuristische Suchverfahren zur Ermittlung möglichst optimaler Netzparameter η verwendet werden. Da diese gradientenfrei sind, benötigen sie auch keine Sensitivitätsinformationen. In der Tat ist dies das Vorgehen zum Training von CTRNN, das in der Literatur bisher am meisten herangezogen wurde (Gallagher u. Vigraham 2002, Falco u. a. 2008, Ahn u. Song 2013). Grundsätzlich kommen für dieses Vorgehen eine Fülle verschiedener Verfahren in Frage.

Diese basieren alle auf der Grundidee, für verschiedene Parameterschätzungen η die CTRNN-Gleichungen zu integrieren und das Ergebnis mit den Trainingswerten zu vergleichen. Davon ausgehend wird dann gemäß einer verfahrenstypischen Heuristik eine neue, nicht zwangsläufig bessere Parameterschätzung ermittelt. Aufgrund der ausbleibenden Nutzung von Gradienteninformationen ist bei diesen Verfahren die Gefahr, nur ein lokales Minimum zu finden, wesentlich geringer als bei den gradientenbasierten Algorithmen (Papageorgiou 2006). Ein gravierender Nachteil dieser Verfahren ist allerdings die Tatsache, dass aufgrund des suchenden Ansatzes Auswertungen vieler verschiedener Parametersätze nötig sind, was wegen der notwendigen Integration der CTRNN-Gleichungen – besonders bei großen Netzen – rechenintensiv werden kann (Kelley 1999).

Ein prominenter Vertreter dieser Verfahren ist beispielsweise das *Simplex*-Verfahren nach Nelder und Mead, bei welchem ein Vieleck durch den Parameterraum wandert, wobei es sich nach gewissen Regeln verformt und zusammenzieht, bis ein Minimum gefunden ist (Nelder u. Mead 1965). Dieses Verfahren ist für seine Robustheit bekannt (Kelley 1999). Ein weiterer metaheuristischer Algorithmus ist die sogenannte *direkte Suche*, bei der systematisch der Parameterraum in gewissen Abständen durchsucht wird, wobei in vielversprechenden Regionen die Testpunkte entsprechend enger gewählt werden. Hierzu eng verwandte Verfahren sind die *Mustersuchverfahren* (engl. *pattern search*), welche die Testpunkte nach gewissen geometrischen Regeln auswählen (Kelley 1999).

Ein weiterer, sehr bekannter metaheurstischer Ansatz zur Minimierung von Kostenfunktionen sind auch die sogenannten *genetischen Algorithmen*, welche durch die biologische Evolution von Lebewesen motiviert sind. Im Gegensatz zu den bisher genannten Verfahren ist in jeder Iteration eine Menge an Lösungskandidaten, die sogenannte Population, vorhanden (Mitchell 1996, Gallagher u. Vigraham 2002). In jeder Iteration können die Elemente dieser Population sich beispielsweise durch Selektion und Rekombination miteinander zu neuen Elementen vereinigen. Hierdurch ist sichergestellt, dass gute Parameterschätzungen erhalten bleiben und ihre Informationen weitergeben können. Darüber hinaus ist auch eine zufällige Mutation vorgesehen, bei der eine Parameterschätzung in der Population sich spontan verändert. Hierdurch kann gewährleistet werden, dass prinzipiell der gesamte Parameterrraum abgesucht werden kann (Mitchell 1996). Genetische Algorithmen sind sehr populär und wurden für eine Vielzahl

von Problemen erfolgreich eingesetzt, darunter auch zum Training von CTRNNs (Li u. a. 2005, Gallagher u. Vigraham 2002, Falco u. a. 2008, Al Seyab 2006).

Aufgrund von Platzbeschränkungen kann an dieser Stelle nicht in gebührendem Detailreichtum auf die einzelnen Verfahren eingegangen werden. Dies ist für die im Rahmen dieser Arbeit behandelten Probleme auch nicht notwendig, da sämtliche genannte Verfahren bereits in implementierter Form verfügbar sind. Wie diese Implementierungen in das Gesamtverfahren eingegliedert werden, wird in Kapitel 4 detailliert erläutert.

3.1.4 Sicherstellung der Modellqualität

Eine essentielle Anforderung an ein neuronales Netz, welches zur Black-Box-Modellierung eines realen Prozesses eingesetzt wird, ist, dass es nach einem erfolgten Training nicht nur die Trainingsdaten zuverlässig wiedergeben kann, sondern – weit wichtiger – auch für neue, unbekannte Eingangsdaten einen Output erzeugt, der möglichst gut mit dem realen System übereinstimmt. Diese wünschenswerte Eigenschaft wird *Generalisierung* genannt und ist bestimmt durch eine angemessene Wahl der Komplexität des neuronalen Netzes, also letztenendes der freien, trainierbaren Parameter. Generell gilt: Je mehr Freiheitsgrade (also Parameter) ein neuronales Netz hat, desto mehr kann es sich den vorgegebenen Trainingsdaten anpassen. Dementsprechend kann theoretisch der Vorhersagefehler des neuronalen Netzes in Bezug auf die Trainingsdaten beliebig weit gesenkt werden (Horvath 2003). Dies ist in der Praxis jedoch nicht wünschenswert, denn je stärker sich das Netz den Trainingsdaten anpasst, desto schlechtere Vorhersagen liefert es für Daten, die nicht im Trainingsdatensatz enthalten waren. In diesem Fall hat das neuronale Netz offenbar eine geringe Generalisierungsfähigkeit und ist als Black-Box-Modell des realen Systems nur in einem sehr eingeschränkten Bereich geeignet.

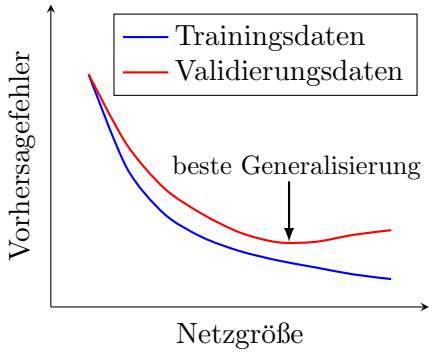
Aus diesem Grund wendet man bei dem Erstellen und Trainieren neuronaler Netze bevorzugt die sogenannte *Kreuzvalidierung* an. Hierunter versteht man das Überprüfen der Modellqualität eines trainierten neuronalen Netzes mit mindestens einem Validierungsdatensatz, welcher nicht im Trainingsdatensatz enthalten ist. Der Vorhersagefehler in Bezug auf die Validierungsdaten stellt ein geeignetes Maß für die Generalisierung des neuronalen Netzes dar. Davon ausgehend kann eine geeignetere Netzgröße gefunden und das neuronale Netz mit modifizierter Topologie erneut trainiert werden um eine bessere Generalisierung zu erreichen.

Dieser Vorgang ist in Abbildung 3.5a dargestellt. Es wird deutlich, dass erwartungsgemäß mit steigender Netzgröße zunächst die Vorhersagefehler im Trainings- sowie im Validierungsdatensatz stark abnehmen. Ab einer gewissen Netzgröße jedoch beginnt der Validierungsfehler wieder zu steigen, während der Trainingsfehler weiter abnimmt. Offenbar ist die Netzgröße mit der besten Generalisierung bestimmt durch das Minimum im Vorhersagefehler des Validierungsdatensatzes.

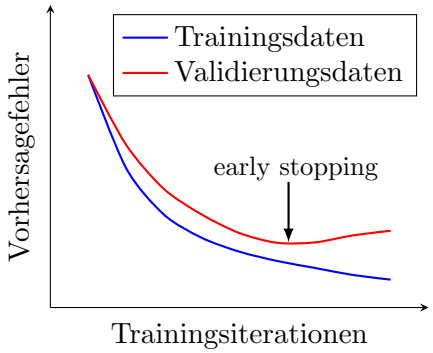
Grundsätzlich ist bei der Wahl einer geeigneten Netzgröße auch die Menge der zur Verfügung stehenden Trainingsdaten zu berücksichtigen. Je weniger Trainingspunkte zur Verfügung stehen, desto weniger Wissen lässt sich daraus in das Netz übertragen und desto weniger Parameter lassen sich somit anpassen. Bei den Problemstellungen, die im Rahmen dieser Arbeit bearbeitet werden sollen, spielt dieser Sachverhalt jedoch eine untergeordnete Rolle, da im Regelfall die Trainingsdaten aus Simulationen stammen werden und somit prinzipiell in nahezu beliebiger Menge erzeugt werden können.

Die iterative Bestimmung einer angemessenen Netzkomplexität, wie sie mit der Kreuzvalidierung durchgeführt wird, hat jedoch den gravierenden Nachteil, dass unter Umständen viele verschiedene Netze mit den gleichen Trainingsdaten trainiert werden müssen. Je nach Lernaufwand, der stark von der Menge der Trainingsdaten und der Netzkomplexität abhängt, kann dieser Prozess sehr zeitaufwändig werden. Daher wurden einige Verfahren vorgeschlagen, um möglichst ohne großen Aufwand die nötige Netzkomplexität abzuschätzen.

Ein wichtiger Vertreter dieser Verfahren ist der sogenannte Lipschitz-Index, welcher die Abschätzung der notwendigen Delay-Werte d_u und d_y allein auf Basis der Trainingsdaten erlaubt (He u. Asada 1993,



(a) Verlust an Generalisierung durch zu hohe Netzkomplexität



(b) Verlust an Generalisierung durch Overfitting/Overtraining

Abbildung 3.5.: Wichtige Phänomene beim Training von neuronalen Netzwerken

Horvath 2003). Dies ist sehr vorteilhaft, da somit nur noch eine geeignete Anzahl an Neuronen n_h in der versteckten Schicht mittels Kreuzvalidierung iterativ ermittelt werden muss. Der Lipschitz-Index beruht auf der Aussage des Lipschitz-Theorems, nach dem stetige Funktion einen beschränkten Gradienten hat, der durch den Gradienten an bekannten Punkten, in diesem Fall den Trainingspunkten, abgeschätzt werden kann (Sragner u. Horvath 2003). Aus dieser Grundidee ergibt sich folgendes Vorgehen. Zunächst wird für alle infrage kommenden Kombinationen (d_u,d_y) in steigender Ordnung der Lipschitz-Quotient nach der Vorschrift

$$L_{i,j}^{(d_y,d_u)} = \|y_j - y_i\| \Big/ \sqrt{\frac{1}{d_y \cdot b} \sum_{p=1}^{d_y} \|y_{j-p} - y_{i-p}\|^2 + \frac{1}{(d_u+1) \cdot a} \sum_{q=0}^{d_u} \|u_{j-q} - u_{i-q}\|^2} \tag{3.20}$$

gebildet (Sragner u. Horvath 2003). Hierin sind $a = \dim(u)$ und $b = \dim(y)$. Mit y_i und u_i werden die Werte von y oder u im Trainingsdatensatz zum i-ten Zeitschritt bezeichnet. Aus den Lipschitz-Quotienten nach Gleichung (3.20) wird dann die Lipschitz-Zahl nach

$$L^{(d_y,d_u)} = \left(\prod_{k=1}^{c} \sqrt{d_y \cdot b + (d_u+1) \cdot a} \cdot L^{(d_y,d_u)}(k) \right)^{\frac{1}{c}} \tag{3.21}$$

berechnet (Sragner u. Horvath 2003). In Gleichung (3.21) bezeichnet $L^{(d_y,d_u)}(k)$ den k-größten Lipschitz-Quotienten unter allen $L_{i,j}^{(d_y,d_u)}$, $i \neq j$. Der Wert c wird in der Literatur mit $c = (0.01 \ldots 0.02) \cdot N$ angegeben (Horvath 2003). Die Lipschitz-Zahl hat nun die Eigenschaft, dass sie für eine gewisse Modellkomplexität (repräsentiert durch d_y und d_u) in einen Sättigungsbereich gerät (Horvath 2003). Das bedeutet, dass die Lipschitz-Zahl optimaler Modellkomplexität wesentlich kleiner als die Lipschitz-Zahl der nächstkleineren Komplexität und etwa so groß wie die der nächstgrößeren Komplexität ist. Diesen Sachverhalt verdeutlicht Abbildung 3.6, welche einen typischen Verlauf der Lipschitz-Zahlen darstellt.

Obwohl die Methode des Lipschitz-Index rechenzeiteffizient und a priori einsetzbar ist, was sie sehr attraktiv macht, so weist sie doch einen gravierenden Nachteil bei der praktischen Anwendung auf: Die Lipschitz-Zahlen verfügen über eine hohe Sensitivität gegenüber Rauschen in den Trainingsdaten (Horvath 2003). In diesem Fall wird der charakteristische Sättigungsbereich der Kurve in Abbildung 3.6 derart verschmiert, dass kein eindeutiges Ablesen der richtigen Delay-Werte mehr möglich ist. Für Probleme, die im Rahmen dieser Arbeit behandelt werden sollen, ist dieser Umstand jedoch nur von geringer Bedeutung, da die Trainingsdaten Simulationen entstammen und somit für gewöhnlich rauschfrei sind.

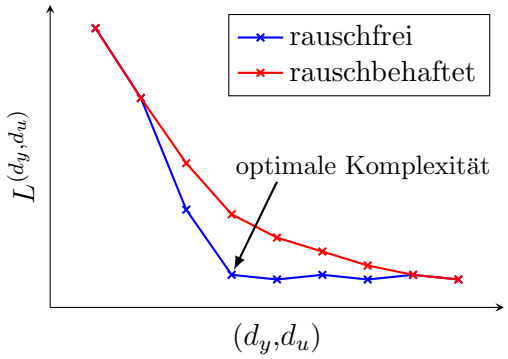


Abbildung 3.6.: Typischer Verlauf der Lipschitz-Zahl nach Gleichung (3.21) für steigende Komplexitätsgrade von NARX-Netzwerken für rauschfreie und rauschbehaftete Trainingsdaten (nach He u. Asada 1993).

Ein weiterer wichtiger Aspekt beim Training qualitativ guter neuronaler Netze ist das Vermeiden von sogenanntem *Overlearning* beziehungsweise *Overfitting* (Horvath 2003). Ähnlich wie bei dem weiter oben angedeuteten Aspekt der richtigen Wahl der Netzgröße ist es ebenso wichtig, den iterativen Trainingsvorgang zu einem geeigneten Zeitpunkt zu beenden. Dies liegt darin begründet, dass neuronale Netze mit einer gewissen Anzahl an freien Parametern den Trainingsdatensatz beinahe beliebig gut erlernen können, wenn nur genug Trainingsiterationen durchgeführt werden. Hierunter leidet jedoch die Generalisierungsfähigkeit des Netzes deutlich, da es die Trainingspunkte bloß mehr und mehr memorisiert und somit keine zufriedenstellende Vorhersagen mit unbekannten Eingangsdaten liefert (Horvath 2003). Aus diesem Grund wird meist wiederum Kreuzvalidierung eingesetzt. Ähnlich wie im weiter oben diskutierten Fall wird während des Trainingsvorgangs stets der Vorhersagefehler in Bezug auf einen Validierungsdatensatz kontrolliert. Die zwei separierten Datensätze werden meist durch ein einfaches Aufspalten der zur Verfügung stehenden Datensätze gewonnen. Der Verlust an Generalisierung durch Overlearning kann verhindert werden, wenn der Trainingsvorgang an dem Punkt gestoppt wird, wo eine weitere Reduktion im Trainingsfehler keine weitere Reduktion im Validierungsfehler mehr mit sich bringt (Horvath 2003). Diese Technik wird *early stopping* genannt und ist in Abbildung 3.5b illustriert. Ein begünstigender Faktor für Overfitting ist eine hohe Anzahl an freien Parametern im neuronalen Netz, da dieses dann die Trainingsdaten wesentlich besser memorisieren kann und weniger zur intelligenten Interpolation zwischen erlernten Punkten gezwungen ist (Horvath 2003).

Es gibt praxisorientierte Hinweise darauf, welcher Anteil der gesamten zur Verfügung stehenden Trainingsdaten zur Bildung eines Validierungsdatensatzes zwecks Kreuzvalidierung abgespalten werden sollte. Wenn das neuronale Netz über eine große Anzahl an Parametern p verfügt, dann ist $(2\sqrt{n_\eta})^{-1}$ eine gute Richtlinie für den Anteil des Validierungsdatensatzes an der gesamten Datenbasis. Für eine Datenbasis mit sehr vielen Elementen, $N > 30n_\eta$ kann aus theoretischer Sicht unter Umständen sogar ganz auf eine Kreuzvalidierung verzichtet werden, da hier keine Möglichkeit für Overfitting mehr gegeben ist (Horvath 2003).

3.1.5 Qualitätsanforderungen an den Trainingsdatensatz

Neben der eigentlichen Topologie eines neuronalen Netzes ist die Qualität der dem Lernprozess zugrunde gelegten Daten von entscheidender Wichtigkeit. Um ein hochwertiges Netz zu erhalten, das einen Prozess über einen breiten Bereich von Betriebszuständen abzubilden vermag, sollte dieser Datensatz also einer Reihe von Anforderungen genügen, die im Folgenden dargelegt werden.

Eine der wichtigsten Bedingungen, die an eine Datenbasis zu stellen ist, ist die Repräsentativität der Daten in Bezug auf den aktuellen Prozess. Das bedeutet insbesondere, dass die Trainingsdaten alle möglichen Zustände und Eingangsgrößen möglichst unabhängig voneinander beinhalten sollte. Dabei

sollten die Eingangsgrößen uniform verteilt sein um dem neuronalen Netz in gleichem Maße Informationen über alle Betriebsbereiche zugänglich zu machen. Hierdurch sind im Trainingsprozess die Wirkungen der einzelnen Inputs voneinander zu isolieren und ein adäquates Modell erlernbar (Horvath 2003). Während diese Anforderung bei Messdaten von realen, produktiv ablaufenden Prozessen oftmals nicht erfüllt wird, so kann dies bei der in dieser Arbeit entwickelten Vorgehensweise beim simulationsgestützten Erzeugen von Trainingsdaten gewährleistet werden.

Ein weiteres wichtiges Kriterium für eine gute Datenbasis ist darüber hinaus auch die bloße Datenmenge. Generell gilt, dass ein neuronales Netz umso besser trainiert werden kann, je mehr (hochwertige) Trainingsdaten es geliefert bekommt. Netze mit vielen freien Parametern benötigen tendenziell auch eine größere Datenbasis für ein erfolgreiches Training (Horvath 2003). Einer Datenbasis mit sehr vielen Datenpunkten steht allerdings entgegen, dass der Lernprozess zunehmend mehr Rechenzeit in Anspruch nimmt. Dies kann den Lernaufwand schnell vervielfachen, da neuronale Netze üblicherweise nicht nur einmal sondern mehrfach trainiert werden, um stochastische Effekte im Lernprozess auszugleichen. Da im Rahmen dieser Arbeit die Trainingsdaten simulationsgestützt erzeugt werden, ist die Datenmenge grundsätzlich frei wählbar, sodass ein angemessener Umfang der Datenbasis eingehalten werden kann.

Abgesehen von der bloßen Länge der Trainingsdaten ist die Dimension derselben jedoch auch wesentlich für deren Umfang. So kann es bei sehr komplexen Prozessen nötig sein, mehrere Eingangsgrößen und Ausgangsgrößen in den Trainingsdatensatz aufzunehmen. Aufgrund der vorstehend genannten Bedingung der Repräsentativität müssen die Eingangsdaten die gesamte Bandbreite möglicher Steuerungen abdecken. Dies hat eine schwerwiegende Konsequenz: Ist n_u die Anzahl der Eingangsgrößen eines zu modellierenden Prozesses und R die diskreten Intervalle, in die der Wertebereich dieser Steuerungen unterteilt wird, so folgt für die Menge möglicher Kombinationen von Steuerungen R^{n_u} (Horvath 2003). Aufgrund dieses exponentiellen Zusammenhangs, der den Umfang der nötigen Trainingsdaten bei relativ komplexen Prozessen überproportional erhöht, wird bei diesem Problem oftmals vom berühmten *Fluch der Dimensionalität (*engl. *curse of dimensionality)* gesprochen. Dieser Begriff wurde von Richard Bellman im Bereich des maschinellen Lernens und der dynamischen Programmierung[3] geprägt, ist jedoch auch im hier diskutierten Zusammenhang von Bedeutung (Bellman 1961). Um dieses Problem zu umgehen oder zumindest zu reduzieren, sollte also im Rahmen einer Modellreduktion bereits in der Konzeption eines neuronalen Netzes die Dimensionalität der Steuerungen möglichst klein gehalten werden.

Darüber hinaus ist auch die Rauschfreiheit der Trainingsdaten eine wichtige Eigenschaft für den Lernprozess. Im Allgemeinen erhöhen verrauschte Daten die Gefahr von Overfitting, da ein zu groß gewähltes neuronales Netz zum Memorisieren des Rauschens neigt (Haykin 1999). Dieses Kriterium ist hier jedoch ebenfalls von untergeordneter Bedeutung, da wie bereits erwähnt die Trainingsdaten im Rahmen dieser Arbeit Simulationen entstammen und somit rauschfrei sind.

Von großer Bedeutung für den Lernprozess eines neuronalen Netzwerks ist eine Konditionierung der Trainingsdaten. In diesem Zusammenhang wird für gewöhnlich eine Normalisierung der Datenbasis vorgenommen, bei welcher alle Daten linear auf das Intervall $[-1,1]$ oder $[0,1]$ skaliert werden (Sola u. Sevilla 1997). Dieses Vorgehen ist besonders wichtig, wenn Werte in den Trainingsdaten unterschiedliche Größenordnungen haben. Bei der Modellbildung von physikalischen Prozessen tritt dieser Fall sehr häufig auf, insbesondere dann, wenn hydraulische Drücke oder mechanische Spannungen, welche in der Größenordnung 10^5 bis 10^8 vorliegen können, Teil der Problemgrößen sind. Derartige Situationen wirken sich negativ auf den Trainingsprozess aus, was unter anderem folgende Gründe hat:

- Die meisten Lernalgorithmen für neuronale Netze initialisieren alle Netzparameter mit einem von Null verschiedenen Zufallswert zwischen -1 und 1 (Demuth u. a. 2009).

- Die Steigung der meistbenutzten Aktivierungsfunktionen, insbesondere der tanh-Funktion, ist außerhalb ihrer Sättigung von der Größenordnung 1. Dies gilt auch für den Wertebereich dieser Funktionen.

[3] Hierbei handelt es sich um theoretische Ansätze in der mathematischen Optimierung, welche sich parallel zu den in Abschnitt 3.2 vorgestellten Verfahren entwickelt haben und beispielsweise in der Robotik vielfach Anwendung finden.

- Bei ausbleibender Normalisierung haben die großskaligen Problemgrößen einen entscheidenden Anteil am Vorhersagefehler. Da dieser eine essentielle Rolle in allen Lernverfahren spielt, können Vorhersagefehler in kleinskaligen Größen ohne Normalisierung nicht hinreichend berücksichtigt werden (Horvath 2003).

In systematischen Studien wurde zudem herausgefunden, dass eine adäquate, lineare Normalisierung neben der Abwendung dieser Nachteile einerseits den Vorteil hat, dass geringere Vorhersagefehler erreichbar sind und andererseits der Trainingsvorgang selbst um etwa eine Größenordnung beschleunigt wird (Sola u. Sevilla 1997).

3.2 Optimalsteuerungsprobleme und ihre Lösung

Nachdem im vorigen Kapitel neuronale Netzwerke zur Bildung von Black-Box-Ersatzmodellen ausführlich eingeführt worden sind, soll in diesem Abschnitt die Theorie der optimalen Steuerungen und entsprechender numerischer Lösungsverfahren dargelegt werden.

3.2.1 Mathematische Problemformulierung

Optimierungsprobleme, die nach einer optimalen Steuerung eines zeitabhängigen physikalischen Prozesses fragen, wurden bereits in der Einleitung sowie Abschnitt 2.2 dieser Arbeit kurz thematisiert. An dieser Stelle soll deren Formulierung noch einmal aufgegriffen und vertieft werden. Die folgenden Ausführungen beziehen sich auf Systeme von autonomen, gewöhnlichen Differentialgleichungen erster Ordnung. Grundsätzlich lassen sich die hier präsentierten Erkenntnisse jedoch auf nichtautonome Differentialgleichungssysteme übertragen, die zudem auch eine höhere Ordnung aufweisen können (Papageorgiou 2006). Darüber hinaus können die Erkenntnisse unter gewissen Bedingungen auch auf allgemeine, implizite, differential-algebraische Gleichungssysteme der Form $F(x(t), \dot{x}(t), u(t), t) = 0$ erweitert werden. Für entsprechende Details sei auf die Spezialliteratur verwiesen (Pinho u. Vinter 1997).

In Anlehnung an die in Abschnitt 2.2 eingeführte Notation lässt sich das allgemeine Optimalsteuerungsproblem wie folgt formulieren (Kirk 1970):

$$\min_{u \in U} \quad J[u(t)] = \phi(x(t_f), t_f) + \int_0^{t_f} L(x(t), u(t))\,\mathrm{d}t \tag{3.22a}$$

$$\text{u.B.v.} \quad \dot{x}(t) = f(x(t), u(t), t) \tag{3.22b}$$

$$g(x(t), u(t), t) \geq 0 \tag{3.22c}$$

Hierin sind $g \in \mathbb{R}^{n_g}$ der Vektor der Zustands- und Steuerungsbeschränkungen, die durch Ungleichungen repräsentiert werden. Gleichung (3.22b) repräsentiert die Systemdynamik. Der sogenannte MAYER-Term ϕ und der LAGRANGE-Term L im Kostenfunktional können auf einfache Weise durch Einführen neuer Systemzustände ineinander überführt werden. Gesucht ist eine zeitabhängige Steuerung $u(t)$, die gemäß der Beschränkungen (3.22c) in der zulässigen Menge U enthalten ist und das Kostenfunktional (3.22a) minimiert. Randbedingungen an $x(0)$ und $x(t_f)$ können entweder vorgegeben oder frei sein, solange die Integration des Differentialgleichungssystems ein wohlgestelltes Problem bleibt. Darüber hinaus kann die Endzeit t_f frei und Teil des Optimierungsergebnisses sein.

3.2.2 Notwendige Optimalitätsbedingungen

Mithilfe der Theorie der Variationsrechnung lassen sich notwendige Bedingungen[4] für eine Steuerung $u(t)$ herleiten, die das Gütefunktional (3.22a) unter den Beschränkungen (3.22b) minimiert.

Hierzu wird zunächst die sogenannte HAMILTON-Funktion durch Ankoppeln der Nebenbedingungen an die Gütefunktion $L(x,u)$ erzeugt (Papageorgiou 2006, Kirk 1970):

$$H(x,u,\lambda,\eta) := L(x,u) + \lambda^T(t) \cdot f(x,u) + \eta^T(t) \cdot g(x,u) \tag{3.23}$$

Mit der auf diese Weise definierten HAMILTON-Funktion lauten die notwendigen Optimalitätsbedingungen für das Optimalsteuerungsproblem (3.22a)–(3.22c) unter der Annahme von genügend differenzierbaren Problemfunktionen, einer regulären HAMILTON-Funktion mit einem eindeutigen Minimum sowie einer autonomen Problemformulierung wie folgt (Stryk u. Bulirsch 1992, Stryk 1993; 1995):

[4] Diese Bedingungen sind *nicht* hinreichend: Eine optimale Steuerung erfüllt die nachfolgenden Bedingungen, aber eine Steuerung, die die Bedingungen erfüllt, ist nicht notwendigerweise optimal.

- Kanonische Differentialgleichungen für den Zustandsvektor $x(t)$ und die sogenannten *adjungierten Variablen* $\lambda(t)$:

$$\dot{x}_i(t) = \frac{\partial H}{\partial \lambda_i} = f_i(x,u) \tag{3.24}$$

$$\dot{\lambda}_i(t) = -\frac{\partial H}{\partial x_i} = -\sum_{k=1}^{n} \lambda_k(t) \cdot \frac{\partial f_k(x,u)}{\partial x_i} - \sum_{j=1}^{n_g} \eta_j(t) \cdot \frac{\partial g(x,u)}{\partial x_i} \tag{3.25}$$

- Für die Multiplikatorfunktion $\eta(t)$ in der HAMILTON-Funktion (3.23) gilt:

$$\eta(t) \begin{cases} = 0 & \text{falls } g(x,u) > 0 \\ \leq 0 & \text{falls } g(x,u) = 0 \end{cases} \tag{3.26}$$

- Das bekannte *Maximumprinzip* besagt, dass eine optimale Steuerung u die HAMILTON-Funktion minimiert:

$$H(x,u,\lambda,\eta) = \min_{\bar{u} \in U} H(x,\bar{u},\lambda,\eta) \tag{3.27}$$

  Wenn im Speziellen H regulär ist, u nichtlinear in f vorkommt und unbeschränkt (d.h. U offen) ist, gilt anstatt Gleichung (3.27) vereinfachend:

$$\frac{\partial H}{\partial u_j} = \sum_{k=1}^{n} \lambda_k(t) \cdot \frac{\partial f_k(x,u)}{\partial u_j} + \sum_{j=1}^{n_g} \eta_j(t) \cdot \frac{\partial g(x,u)}{\partial u_j} = 0 \tag{3.28}$$

- Für Zustände x_i, deren Anfangs- oder Endzustand nicht im Optimierungsproblem (3.22a)–(3.22c) gegeben ist, ist der Anfangs- beziehungsweise Endwert des zugehörigen adjungierten Zustands λ_i wie folgt gegeben:

$$\lambda_i(0) = 0 \tag{3.29}$$

$$\lambda_i(t_f) = \frac{\partial \phi(x(t_f), t_f)}{\partial x_i(t_f)} \tag{3.30}$$

- Darüber hinaus gilt im Falle eines Problems mit freier Endzeit die zusätzliche Randbedingung

$$H(x(t_f), u(t_f), \lambda(t_f), \eta(t_f)) = -\frac{\partial \phi(x(t_f), t_f)}{\partial t_f} \tag{3.31}$$

- Da die Problemstellung autonom ist, ist die HAMILTON-Funktion abschnittsweise konstant. Sind zusätzlich keine Ungleichungsbeschränkungen $g(x,u)$ vorgegeben, ist die HAMILTON-Funktion auf dem ganzen Zeitintervall $[0,t_f]$ konstant:

$$H(x(t), u(t), \lambda(t), \eta(t)) = \text{const.} \tag{3.32}$$

- An Schaltpunkten t_s, in denen Zustands- oder Steuerungsbeschränkungen ihre Aktivität wechseln, sind die adjungierten Zustände λ im Allgemeinen unstetig und es gilt der vektoriell aufzufassende Zusammenhang

$$\lambda(t_s^+) = \lambda(t_s^-) - \nu_s \nabla_x g(x(t_s),u(t_s)) \tag{3.33}$$

  mit $\nu_s \leq 0$. Die Notation t_s^+ bezeichnet den Zeitpunkt unmittelbar nach der Umschaltung und t_s^- entsprechend unmittelbar davor.

Die vorstehenden Bedingungen bilden die Grundlage für die Ermittlung der optimalen Steuerung u. Welche numerischen Verfahren hierfür existieren, in wie fern sie auf die notwendigen Bedingungen zurückgreifen und welche Verfahren für die im Rahmen dieser Arbeit behandelten Probleme vorteilhaft sind, wird im Abschnitt 3.2.4 ausführlich dargelegt.

3.2.3 Formulierungen verschiedener Kostenfunktionen

Beim Aufstellen des Optimierungsproblems nach den Gleichungen (3.22a)–(3.22c) kann durch die konkrete Wahl des MAYER-Terms $\phi(x(t_f), t_f)$ beziehungsweise des LAGRANGE-Terms $L(x,u)$ physikalische Sinnhaftigkeit in das Gütefunktional eingeführt werden. Dies ermöglicht das Durchführen der Optimierung hinsichtlich verschiedener Vorgaben, die in der Praxis einander durchaus entgegenstehen können. Dabei ist zu beachten, dass sich der MAYER-Term ϕ durch Interpretation als zusätzlichen Systemzustand in LANGRANGE-Form transformieren lässt. Daraus folgt, dass die folgende exemplarische Auflistung möglicher Kostenfunktionen in LAGRANGE-Form ohne Beschränkung der Allgemeinheit in Gleichung (3.22a) sind. Wichtige Kostenfunktionen sind nun die Folgenden (Kirk 1970):

Zeitminimalität verlangt, dass der Prozess, der dem Differentialgleichungssystem (3.22b) genügt, möglichst schnell unter den angegebenen Beschränkungen (3.22c) von einem spezifizierten Anfangs- in einen spezifizierten Endzustand gebracht werden soll. Diese Forderung lässt sich mathematisch wie folgt modellieren:

$$L(x,u) = 1 \tag{3.34}$$

Verbrauchsoptimalität kann gefordert werden, um den Prozess möglichst sparsam in Bezug auf den Steuerungseinsatz in einen bestimmten Endzustand zu überführen. Die entsprechende LANGRANGE-Kostenfunktion lautet dann:

$$L(x,u) = |u| \tag{3.35}$$

Ein Nachteil an dieser Formulierung ist die Betragsbildung, die eine Nichtdifferenzierbarkeit in das Optimierungsproblem einführt.

Energieminimalität bewertet, ähnlich wie eine Forderung nach Verbrauchsoptimalität, den notwendigen Steueraufwand, um einen Prozess in einer vorgegebenen Zeit in einen gewissen Endzustand zu überführen:

$$L(x,u) = u^2 \tag{3.36}$$

Durch eine Quadrierung des Steueraufwands werden aggressive Steuerungsstrategien stärker bestraft als milde Steuerungen.

Kombinierte Zeit- und Energieminimalität hat besonders in der Praxis eine hohe Bedeutung. In vielen Situationen, insbesondere im Betrieb verfahrenstechnischer Anlagen, muss ein Kompromiss zwischen einer möglichst schnellen und einer – in welcher Form auch immer – sparsamen Steuerung gefunden werden. Derartige Begebenheiten lassen sich im Kostenfunktional durch eine gewichtete Kombination von Zeit- und Energieminimalität wie folgt berücksichtigen:

$$L(x,u) = \omega + u^2 \tag{3.37}$$

Auf analoge Weise lässt sich natürlich auch eine Kombination von Zeit- und Verbrauchsoptimalität heranziehen. In Gleichung (3.37) ist nur ein Gewichtungsfaktor ω entweder für die Zeit- oder die Energieminimalität notwendig, da im Allgemeinen das Optimierungsergebnis unabhängig von einer Skalierung des Kostenfunktionals mit einer Konstante ist (Kirk 1970).

Bei den hier präsentierten Formulierungen verschiedener LANGRANGE'scher Kostenfunktionen gilt es zu beachten, dass problemabhängig eine Integration von $|u|$ oder u^2 nicht unbedingt eine physikalische Größe hervorbringt, die einen Verbrauch oder eine Energie repräsentiert. Daher ist die physikalische Bedeutung von $u(t)$ immer auf den Einzelfall bezogen zu interpretieren und in der Kostenfunktion entsprechend zu berücksichtigen (Kirk 1970).

Darüber hinaus gilt im Allgemeinen, dass bei wohlgestellten Optimalsteuerungsproblemen die Lösung immer abhängig von der Kostenfunktion ist. Dementsprechend gilt, dass ein Problem mit einer gewissen Kostenfunktion und Beschränkungen eventuell auch nicht lösbar sein kann (Kirk 1970).

3.2.4 Lösung von Optimalsteuerungsproblemen

Bei der Lösung von Optimalsteuerungsproblemen der hier diskutierten Form können ganz unterschiedliche Verfahren zum Einsatz kommen. Zur Einteilung dieser Verfahren hat sich eine Unterscheidung zwischen indirekten und direkten Ansätzen etabliert (Stryk 1995).

Indirekte Verfahren beruhen auf der numerischen Lösung des Gleichungssystems, das durch die Optimalitätsbedingungen nach Abschnitt 3.2.2 gebildet wird. Diese stellen in ihrer Gesamtheit ein sogenanntes *Mehrpunkt-Randwertproblem* dar. Zur numerischen Lösung derartiger Systeme stehen Verfahren wie etwa das Mehrfachschießverfahren zur Verfügung. Dadurch kann das Optimalsteuerungsproblem mit großer Genauigkeit gelöst werden, was zum Beispiel bei Problemen in der Raumfahrt (beispielsweise der optimale Wiedereintritt eines Shuttles in die Erdatmosphäre) wichtig ist (Stryk 1995).

Direkte Verfahren hingegen parametrisieren die Steuerung beispielsweise stückweise konstant oder stückweise linear in Abhängigkeit von einem Satz an Parametern p. Optional können auch die Systemzustände x parametrisiert werden. Hierdurch wird das unendlich-dimensionale Optimierungsproblem in ein endlich-dimensionales Optimierungsproblem überführt, das effizient mit den Verfahren der nichtlinearen, statischen Optimierung gelöst werden kann. Das resultierende endlich-dimensionale Problem, das üblicherweise als *non-linear programming problem* (NLP) bezeichnet wird, hat dann die folgende (vektoriell aufzufassende) Form[5] (Papageorgiou 2006):

$$\min_{p} \quad \varphi(p) \tag{3.38a}$$

$$\text{u.B.v.} \quad \tilde{g}(p) = 0 \tag{3.38b}$$

$$\tilde{h}(p) \geq 0 \tag{3.38c}$$

Zur Theorie von NLPs und deren Lösung existiert eine Fülle an Spezialliteratur, deren Abhandlung außerhalb des Rahmens der vorliegenden Arbeit liegt. Der interessierte Leser sei auf die entsprechende Literatur verwiesen (zum Beispiel Nocedal u. Wright 1999, Papageorgiou 2006).

Indirekte Verfahren haben gegenüber den direkten Verfahren den gravierenden Nachteil, dass deren Anwendung mitunter sehr aufwändige Vorarbeit erfordert. So müssen beispielsweise die notwendigen Bedingungen nach Abschnitt 3.2.2 zunächst in expliziter Weise formuliert werden, was je nach Komplexität des Systems eine erhebliche und fehleranfällige Arbeit bedeutet. Erschwerend kommt hinzu, dass die Schaltpunkte t_s, an denen die vorliegenden Beschränkungen aktiv oder inaktiv werden, a priori nicht bekannt sind. Ebenso problematisch ist, dass zur numerischen Lösung der Optimalitätsbedingungen gute Startschätzungen für die adjungierten Variablen $\lambda(t)$ und $\eta(t)$ notwendig sind, die aufgrund ihrer fehlenden intuitiven Bedeutung nur schwer bestimmbar sind. Demgegenüber steht der Vorteil der indirekten Verfahren, dass sie Lösungen mit höchster Genauigkeit erlauben. Dies ist allerdings für Probleme, wie sie in der vorliegenden Arbeit behandelt werden, von untergeordneter Bedeutung, da ohnehin durch die Bildung von Ersatzmodellen Approximationsfehler gemacht werden, die weitaus bedeutsamer sein dürften als die Fehler in der eigentlichen Lösung des Optimalsteuerungsproblems. Aus diesem wird im Folgenden ein Fokus auf die direkten Verfahren gelegt.

Wichtige Vertreter der direkten Verfahren sind das sogenannte *direkte Schießen* und die *direkte Kollokation* (Stryk 1995). Bei letzterem Ansatz werden sowohl die Steuerung als auch die Systemzustände durch polynomiale Ansatzfunktionen parametrisiert, wobei der Grad des Approximationspolynoms in der Regel für die Systemzustände höher gewählt wird als für die Steuerung (Stryk 1995). Das gesamte Zeitintervall wird in eine Reihe von Zeitsegmenten unterteilt und die polynomialen Ansatzpolynome für jedes

[5] Die Notation in den Gleichungen (3.38a) bis (3.38c) ist unabhängig und abweichend von der bis hierhin verwendeten Notation.

dieser Segmente aufgestellt. Durch die Forderung von Stetigkeit und Differenzierbarkeit am Übergang zweier Zeitsegmente entstehen somit Zwangsbeziehungen der benachbarten Polynome zueinander, sodass einige Parameter in einem gewissen Verhältnis zu anderen Parametern stehen. Diese Zusammenhänge dienen als Gleichungsnebenbedingungen bei der Lösung des entstehenden NLPs. Die ursprünglichen Nebenbedingungen des Optimalsteuerungsproblem finden darüber hinaus ebenfalls in angepasster Form Verwendung im NLP (Stryk 1995).

Um die Ansatzpolynome auf einem Zeitsegment eindeutig festzulegen, ist es nötig, eine dem Polynomgrad entsprechende Menge von Stützpunkten festzulegen. Abhängig davon, wie diese Stützpunkte gewählt werden, ergeben sich unterschiedliche Kollokationsverfahren. Ein wichtiges Kollokationsschema ist beispielsweise die Kollokation mit *Gauß*-Punkten, die drei Stützpunkte verwendet, die im inneren des Zeitsegments liegen. Dieses Schema garantiert zwar die theoretisch bestmögliche Approximation, führt jedoch an den Übergängen zweier Zeitsegmente zu nicht-differenzierbaren Approximationen. Ein alternatives Schema, das dieses Problem umgeht, ist das sogenannte *Lobatto*-Schema, welches zwei der drei Stützpunkte so wählt, dass sie genau auf den Übergängen zwischen den Zeitsegmenten liegen. Dies hat den positiven Nebeneffekt, dass die Anzahl von Stützpunkten auf dem gesamten Zeitintervall verringert wird, weil zwei benachbarte Polynome sich immer einen Stützpunkt teilen (Stryk 1995).

Ein weiterer wichtiger Ansatz der Klasse der direkten Verfahren ist das direkte Schießen. Im Gegensatz zu den Kollokationsverfahren wird hier nur die Steuerung parametrisiert. Der Lösungsprozess des Optimierungsproblems geht dann so vonstatten, dass für einen vorläufigen Satz von Parametern, die die Steuerung bestimmen, die Systemgleichungen integriert werden und so die zur vorläufigen Steuerung gehörigen Systemzustände erhalten werden. Daraus kann dann der Wert des Gütekriteriums und die Verletzung von Beschränkungen ermittelt werden. Diese Zwischenergebnisse werden nun im eigentlichen Orientierungsverfahren verwendet, um einen neuen Satz von Steuerungsparametern zu erzeugen, der das Gütekriterium minimiert und die Einhaltung der Beschränkungen besser gewährleistet. Mit diesem Parametersatz beginnt eine neue Iteration, an deren Anfang wieder die Lösung der Systemgleichungen mit der aktualisierten Steuerung steht (Stryk 1993; 1995).

Ein wichtiger Vorteil des direkten Schießverfahrens gegenüber Kollokationsverfahren ist, dass die Dimension des resultierenden NLPs weitaus kleiner ist, sodass dessen Lösung erheblich einfacher erfolgen kann. Allerdings kann, wenn die Näherung der Steuerung noch weit vom Minimum entfernt ist, die Integration des Systems schlecht konditioniert sein und sogar fehlschlagen (Nocedal u. Wright 1999).

Zur numerischen Lösung von allgemeinen Optimalsteuerungsproblemen wurde in der vorliegenden Arbeit die freie Software JModelica.org verwendet, die auf einem direkten Kollokationsverfahren basiert (Akesson 2007, Akesson u. a. 2009). Das aus der Kollokation entstehende NLP wird in diesem Softwarepaket mittels der ebenfalls freien Softwarebibliothek IPOPT gelöst. Ihrem ausgeschriebenen Namen *interior point optimizer* gemäß das NLP mit einem innere-Punkte-Verfahren (Wächter u. Bieger 2006). Diese Technik zeichnet sich unter Anderem dadurch aus, dass in jeder Iteration des NLP-Lösers eine zulässige Zwischenlösung vorliegt, was beispielsweise bei den berühmten Verfahren der *sequentiellen quadratischen Programmierung* (SQP) nicht der Fall ist. Auf die Einzelheiten der innere-Punkte-Verfahren soll hier jedoch nicht eingegangen werden. Stattdessen sei auf die Spezialliteratur verwiesen (Kelley 1999, Papageorgiou 2006, Wächter u. Bieger 2006). Die Einbindung von IPOPT in das Gesamtverfahren wird ausführlich im folgenden Kapitel 4 beschrieben.

4 Entwicklung und Implementierung des Verfahrens

Nachdem in den vorherigen Abschnitten zunächst ein grobes Verfahrenskonzept skizziert und die notwendigen theoretischen Grundlagen aufgearbeitet wurden, soll nun in diesem Kapitel das Verfahren in seiner Gänze und in gebührendem Detail entwickelt und diskutiert werden. Hierzu wird zunächst im Abschnitt 4.1 eine Differenzierung verschiedener Verfahrensvarianten mit ihren Vor- und Nachteilen vorgenommen und darauf aufbauend in den folgenden Abschnitten 4.2 bis 4.6 die einzelnen Verfahrensschritte ausgearbeitet. Abschließend wird im Abschnitt 4.6 noch eine globale Fehlerdiskussion vorgenommen, um Fehlerquellen des Verfahrens zu identifizieren und diskutieren.

4.1 Verfahrensvarianten

Wie bereits angedeutet sind verschiedene Verfahrensvarianten denkbar, die allesamt aus den grundlegenden Überlegungen der Verfahrenskonzeption nach Kapitel 2 folgen. Diese Varianten lassen sich anhand von Abbildung 4.1, die den generellen Ablauf des Verfahrens aufzeigt, nachvollziehen.

Allen Verfahren gemein ist zunächst, dass aus dem Originalsystem, das sowohl als physisches System oder Simulationsmodell vorliegen kann, Trainingsdatensätze gewonnen werden. Hierbei ist, soweit möglich, auf die Anforderungen an die Qualität dieser Datensätze zu achten, wie in Abschnitt 3.1.5 diskutiert wurde. An dieser Stelle lassen sich nun zweierlei Wege beschreiten: Entweder kann mithilfe der Trainingsdaten ein zeitdiskretes (NARX) oder ein zeitstetiges (CTRNN) neuronales Netz als Ersatzmodell erzeugt werden. Unabhängig von dieser Wahl sollte eine Validierung des neuronalen Netzes mit Validierungsdaten vorgenommen werden, um die Qualität des trainierten neuronalen Netzwerks als Ersatzmodell abzuschätzen.

Wurde zum Erzeugen eines Ersatzmodells ein zeitdiskretes neuronales Netz gewählt, so lässt es sich optional in ein zeitstetiges Netz umwandeln. Der Vorteil dieses Vorgehens ist, dass auch zeitdiskrete Netze einer zeitstetigen Optimierung mittels direkter Kollokation zugänglich gemacht werden. Wird eine derartige Umwandlung nicht vorgenommen, so lässt sich immer noch mit einem entsprechendem zeitdiskret arbeitenden Optimierungsalgorithmus eine Optimierung vornehmen. Hierzu wurde in der vorliegenden Arbeit das direkte Schießen ausgewählt.

Somit lassen sich folgende drei Verfahrensvarianten differenzieren:

1. **DNDO (*discrete net, discrete optimization*)**: Erzeugen eines zeitdiskreten Netzes, optimieren am zeitdiskreten Netz.

2. **DNCO (*discrete net, continuous optimization*)**: Erzeugen eines zeitdiskreten Netzes, optimieren am zeitstetigen Netz.

3. **CNCO (*continuous net, continuous optimization*):** Erzeugen eines zeitstetigen Netzes, optimieren am zeitstetigen Netz.

Welche Art von neuronalem Netz im jeweiligen Anwendungsfall auszuwählen ist, ist in hohem Maße problemabhängig. Im Idealfall sollten beide Varianten ähnlich gute Resultate liefern. Jede Variante bietet jedoch eigene Vorteile, auf die im Abschnitt 4.3 näher eingegangen wird.

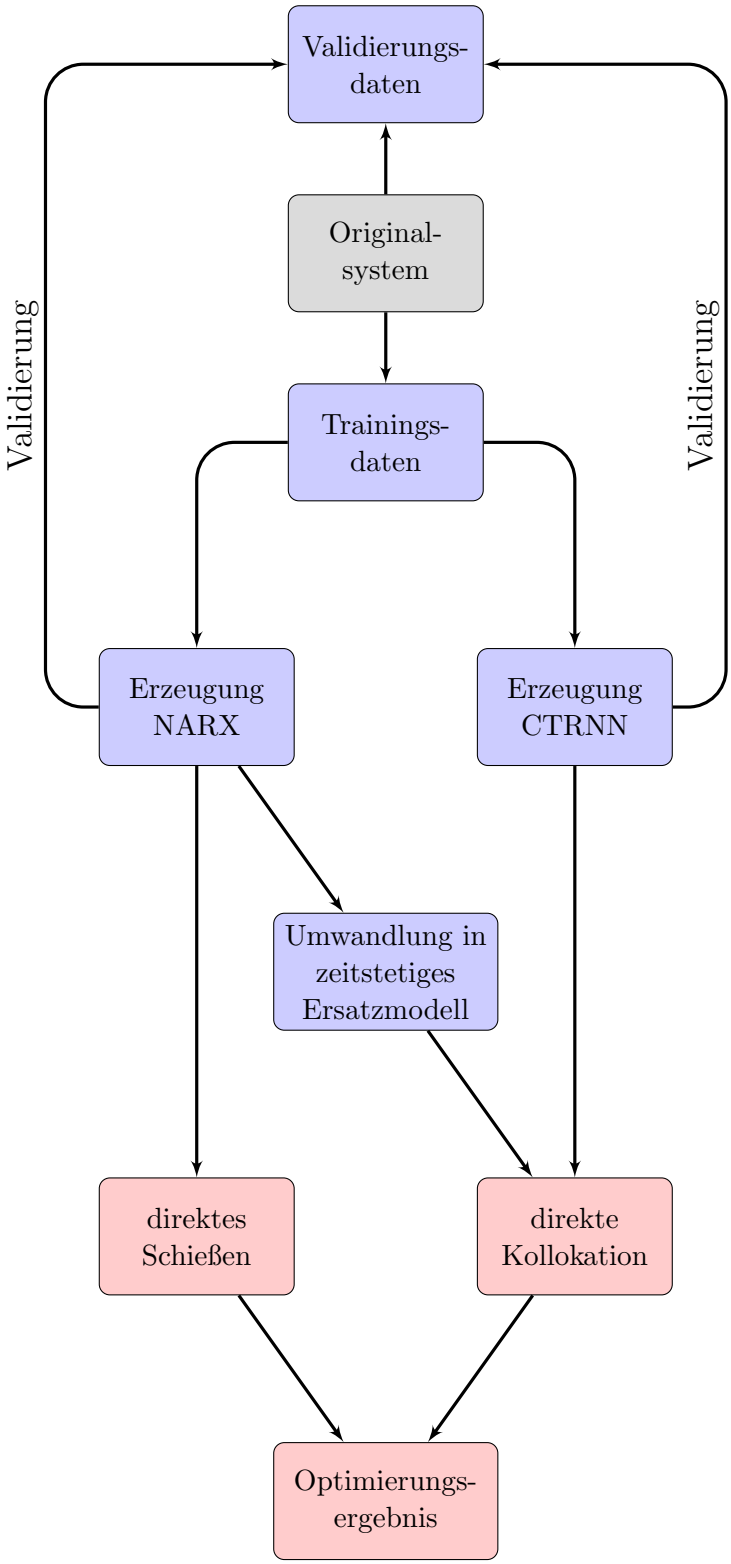


Abbildung 4.1.: Flow-Chart der verschiedenen Verfahrensvarianten.

4.2 Erzeugen der Trainingsdatensätze

Bereits das Ableiten von Trainingsdatensätzen aus dem Originalsystem ist von kritischer Bedeutung und kann entscheidend dafür sein, ob ein qualitativ gutes Ersatzmodell erzeugt werden kann. Dementsprechend ist der Qualität der Trainingsdaten eine hohe Bedeutung beizumessen. Wichtige Anforderungen sind bereits in Abschnitt 3.1.5 formuliert worden und sollten nach Möglichkeit erfüllt werden.

Eine wichtige Bedeutung hat auch die Zeitschrittweite $\triangle t$ der Trainingsdaten. Aus Sicht des Trainingsprozesses sollte sie so klein wie nötig, aber so groß wie möglich sein. Mit einer adäquaten Schrittweite wird das Zeitverhalten des Prozesses noch hinreichend gut aufgelöst, sodass ein Ersatzmodell diese auch erlernen kann. Andererseits sind die Schrittweiten groß genug, um möglichst lange Trainingsdatensätze zu erhalten, ohne auf eine große Anzahl von Stützstellen zurückgreifen zu müssen. Letztere erhöht die Rechenzeit zum Training der neuronalen Netze deutlich. Darüber hinaus spielt die Zeitschrittweite eine wichtige Rolle bei der Umwandlung von einem zeitdiskreten NARX-Netzwerk zu einem zeitstetigen Ersatzmodell. Hierauf wird im entsprechenden Abschnitt 4.4 eingegangen.

Ein Vorteil des im Rahmen dieser Arbeit entwickelten Verfahrens ist, dass eine große Flexibilität bezüglich der Datenquelle besteht. Das bedeutet, dass das Gesamtverfahren nicht nur auf Simulationsmodelle, sondern auf verschiedenste Systeme, so etwa auch auf reale Anlagen, angewendet werden kann.

4.3 Trainieren der neuronalen Netzwerke

Ein wesentlicher Bestandteil des hier entwickelten Verfahrens ist der Trainingsprozess der neuronalen Netzwerke. Ziel des Trainingsprozesses ist das Anpassen der freien Parameter eines neuronalen Netzes derart, dass es das ursprüngliche Systemverhalten bestmöglich wiedergibt. Im Abschnitt 3.1 über die Theorie neuronaler Netzwerke wurde bereits herausgestellt, dass es sich bei derartigen Netzwerken um universale Approximatoren handelt. Fraglich bleibt jedoch, ob dieser Zustand der besten Vorhersagefähigkeit auch im Trainingsprozess auf ein Netzwerk übertragen werden kann. Dies hängt von einer Vielzahl von Faktoren ab, wie etwa

- der gewählten Netztopologie und den Verzögerungen,
- den eingesetzten Lernverfahren sowie
- den Trainings- und Validierungsdaten.

Da die eigentliche Optimierung anhand des Ersatzmodells durchgeführt wird, steht und fällt der Erfolg des Gesamtverfahrens mit dem Ergebnis des Trainingsprozesses. Daher soll dieser im vorliegenden Abschnitt in gebührendem Detail thematisiert werden.

Zunächst ist vor dem eigentlichen Trainingsprozess eine Netztopologie festzulegen. Diese bestimmt die Komplexität des Netzes und somit auch, wie gut sich ein neuronales Netz an die Trainingsdaten anpassen und diese generalisieren kann.

Im Falle von zeitdiskreten NARX-Netzwerken kann mithilfe einiger Methoden a priori eine Abschätzung notwendiger Netzkomplexitäten vorgenommen werden. So kann beispielsweise auf einfache Weise anhand der Autokorrelation von y oder der Kreuzkorrelation zwischen y und u abgeschätzt werden, welche Verzögerungen von statistisch signifikanter Bedeutung für den Zeitverlauf von y sind. Alternativ liefert der Lipschitz-Index in vielen Fällen ein gutes Maß für die notwendigen Verzögerungen. Die Berechnungsweise des Lipschitz-Index wurde bereits in Abschnitt 3.1 dargelegt. Diese wurde beispielsweise in der frei erhältlichen Matlab-Toolbox NNSYSID auf eine effiziente Weise implementiert, sodass für vorliegende Arbeit auf diese Toolbox zurückgegriffen wurde (Norgaard 1997). Ein Matlab-Code für den Aufruf dieser Funktion ist in Anhang A.1 zu finden. Für die notwendige Anzahl von Neuronen in der versteckten Schicht existieren bis dato keine robusten Vorhersagemethoden, weswegen in der praktischen

Anwendung Netze mit unterschiedlich großen versteckten Schichten trainiert werden sollten. Anhand der Vorhersagequalität bezüglich der Trainings- und der Validierungsdaten ist dann zu entscheiden, welche Komplexität für beide Datensätze gute Ergebnisse liefert. In Anbetracht der Tatsache, dass das Netz bei ausreichender Anzahl von Neuronen fast immer die Trainingsdaten sehr genau wiedergeben kann, sollte die Vorhersagequalität bezüglich der Validierungsdaten prioritär behandelt werden.

Während zur a-priori-Abschätzung der Komplexität von NARX-Netzwerken einige Hilfsmittel existieren, so ist die Lage bei zeitstetigen CTRNN-Netzen nicht so deutlich. Hier sind in der Literatur bisher keine geeigneten Verfahren bekannt, die es erlauben, eine ausreichende Größe der Netze vor dem Trainingsvorgang abzuschätzen. Daher müssen in der Praxis auf heuristische Weise verschiedene Netzkomplexitäten getestet und verglichen werden. Gegenüber NARX-Netzen hat man dabei jedoch den Vorteil, dass je nach CTRNN-Form nur zwei oder sogar nur ein Parameter variierten werden müssen. Bei der Form nach den Gleichungen (3.7a) bis (3.7b) sind dies die Anzahl von Neuronen in der versteckten Schicht, n_h, sowie die Anzahl der inneren Zustände des Netzes, n_x. Wird stattdessen ein CTRNN nach den Gleichungen (3.10a) bis (3.10b) verwendet, ist n_x die einzige Größe, die die Komplexität des Netzes bestimmt.

Nach der Festlegung einer Netzkomplexität beginnt der Trainingsprozess. Abhängig davon, ob man ein NARX- oder ein CTRNN-Netz verwendet, ergeben sich abweichende Vorgehensweisen.

Im Falle von NARX-Netzwerken kann auf die *Neural Network*-Toolbox, die Bestandteil von MATLAB ist, zurückgegriffen werden (Demuth u. a. 2009). Hierbei handelt es sich um eine benutzerfreundliche Funktionenbibliothek für neuronale Netzwerke vielfältiger Arten, die auch effiziente Lernalgorithmen bereithält. Unter Anderem enthält die *Neural Network*-Toolbox auch Lernalgorithmen, die auf dem LEVENBERG-MARQUARDT-Verfahren basieren, welches bereits im Abschnitt 3.1 eingeführt wurde und durch die Gleichungen (3.13) bis (3.16) bestimmt ist. Im Umgang mit der besagten Toolbox sind zum Training eines neuronalen Netzes im Wesentlichen die folgenden Schritte mit den genannten Toolbox-Funktionen durchzuführen:

1. Laden der Trainingsdaten und umwandeln derselben in ein von der Toolbox erwartetes Format mit der Funktion `tonndata`
2. Spezifizieren der Verzögerungen und Anzahl der Neuronen in der versteckten Schicht
3. Iteration über eine gewisse Anzahl von Versuchen. Dies ist notwendig, weil jedes Netzwerk bei seiner Initialisierung zufällige Startparameter zugewiesen bekommt, die ein Zufallselement im Lernprozess darstellen (Demuth u. a. 2009). Um statistisch möglichst signifikante Ergebnisse zu erhalten und negative Ausreißer auszuschließen, sollten in einem Trainingslauf mindestens 50 bis 100 Trainingsversuche vorgesehen werden, die wie folgt aufgebaut sind:
   3.1. Erzeugen eines neuen NARX-Netzes der festgelegten Komplexität mittels des `narxnet`-Befehls
   3.2. Blockweises Aufteilen der gesamten Trainingsdaten in einen Trainings- und einen Validierungsteil (`net.divideParam.trainratio` und `net.divideParam.valratio`). Entsprechend den Ausführungen in Abschnitt 3.1 bietet sich hier ein Anteil der Trainingsdaten von 70 bis 95 % an.
   3.3. Spezifizieren LEVENBERG-MARQUARDT-Verfahrens als Trainingsalgorithmus über `net.trainFcn = 'lmtrain';`
   3.4. Initialisieren des NARX-Netzes und der Eingabe- und Ausgabedatensätze mittels des Befehls `preparets`
   3.5. Training des Netzes im offenen Kreis über die Funktion `train`. Im offenen Kreis wird die Ausgangsgröße wird noch *nicht* zurückgeführt, sondern aus den Trainingsdaten entnommen. Dies erleichtert und stabilisiert das Training (Demuth u. a. 2009).

3.6. Schließen des Rückführungskreises mittels `closeloop`.

3.7. Validieren des resultierenden NARX-Netzes.

3.8. Speichern des Netzes und seiner Qualitätsmerkmale, beispielsweise maximale Abweichung und Korrelation.

4. Auswahl des besten Netzes für die weiteren Untersuchungen

Ein MATLAB-Skript zur automatisierten Durchführung dieser Schritte ist dieser Arbeit in Anhang A.1 beigefügt.

Der Trainingsprozess von zeitdiskreten CTRNN-Netzen geht etwas anders von statten. Wie im Rahmen der theoretischen Aufarbeitung derselben in Abschnitt 3.1 bereits diskutiert wurde, erfordert die Anwendung des LEVENBERG-MARQUARDT-Verfahrens die Ermittlung der zeitabhängigen Sensitivitäten der Vorhersage des CTRNN-Netzes bezüglich der Netzparameter, $\frac{\partial \hat{y}}{\partial \eta}$. Im Rahmen dieser Arbeit wird hierzu eine sogenannte interne numerische Differenziation durchgeführt, bei der während der Integration der eigentlichen CTRNN-Gleichungen auch die Sensitivitätsgleichungen (3.17a) und (3.17b) integriert werden. Anschließend kann in Anlehnung an das in Abschnitt 3.1 beschriebene Vorgehen die für das LEVENBERG-MARQUARDT-Verfahren notwendige Jacobi-Matrix des Vorhersagefehlers bezüglich der Netzparameter berechnet werden. Hierzu kommen die Gleichungen (3.18) und (3.19) zum Einsatz (Al Seyab 2006). Damit ergibt sich die grundlegende Struktur dieses Lernverfahrens wie folgt:

1. Laden und gegebenenfalls Aufbereiten der Trainingsdaten

2. Spezifizieren der Topologie des CTRNN-Netzes und eines Startparametersatzes

3. Schleife mit einer festgelegten Anzahl von Iterationen des LEVENBERG-MARQUARDT-Verfahrens

   3.1. Integration der CTRNN- und Sensitivitätsgleichungen mit der aktuellen Parameterschätzung der k-ten Iteration, η_k.

   3.2. Berechnung des Vorhersagefehlers $\epsilon(\eta_k)$ gemäß Gleichung (3.12).

   3.3. Ermitteln der Jacobi-Matrix $J_i(\eta_k)$ für jeden Zeitpunkt i. Diese sind entsprechend Gleichung (3.19) durch die Sensitivitäten zum Zeitpunkt i festgelegt, die ebenfalls Ergebnis der Integration nach Schritt 3.1 sind.

   3.4. Assemblierung der gesamten Jacobi-Matrix $J_\epsilon(\eta_k)$ gemäß Gleichung (3.18).

   3.5. Aktualisierung der Parameterschätzung gemäß Gleichung (3.16) zur Bestimmung von η_{k+1}.

   3.6. Anpassen des Dämpfungsfaktors μ_{k+1} in Gleichung (3.16) für die nächste Iteration.

4. Visualisierung und Auswertung der Ergebnisse

Ein wichtiger Schritt bei der Implementierung des Verfahrens ist die Bildung der partiellen Ableitungen $\frac{\partial f}{\partial \eta}$ und $\frac{\partial f}{\partial \hat{x}}$, die in den Sensitivitätsgleichungen (3.17a) und (3.17b) benötigt werden. Bei der Funktion f handelt es sich um die CTRNN-Gleichungen, das heißt je nach ausgewählter CTRNN-Topologie entspricht dieser den Gleichungen (3.7a) und (3.7b) beziehungsweise (3.10a) und (3.10b). Um diese nicht von Hand zu berechnen und implementieren, was fehleranfällig wäre, wurde im Rahmen dieser Arbeit die frei erhältliche MATLAB-Bibliothek ADIMAT *(Automatic Differentiation for Matlab)* verwendet, das diese Aufgabe automatisiert (Bischof u. a. 2003). Theoretische Grundlage dieser Software ist die sogenannte *Automatische Differenziation.* Dies bezeichnet eine Technik, bei der auf Basis des Quellcodes einer Funktionsauswertung durch eine algorithmische Analyse ein Quellcode erstellt wird, der die Ableitung dieser Funktionsauswertung nach beliebigen Parametern oder Eingabewerten berechnet (Nocedal u. Wright 1999). Hierdurch kann bei der Implementierung des Verfahrens vermieden werden, die

notwendigen Ableitungen händisch oder nur näherungsweise durch finite Differenzenapproximationen bestimmen.

Ein MATLAB-Skript, welches das soeben beschriebene, auf dem LEVENBERG-MARQUARDT-Verfahren beruhende Verfahren zum Training eines CTRNN-Netzes implementiert, ist in Anhang A.1 zu finden.

Abweichend von gradientenbasierten Lernverfahren, wie das LEVENBERG-MARQUARDT-Verfahren eines ist, lässt sich gemäß Abschnitt 3.1 das Training von CTRNN-Netzen auch mit gradientenfreien Verfahren durchführen. Da diese ihrer Natur gemäß reine Suchverfahren sind, die ihre Suchrichtungen nach gewissen Heuristiken auswählen, ohne jedoch Ableitungsinformationen zu nutzen, sind diese Verfahren im Allgemeinen weniger performant als die gradientenbasierten Lernverfahren (Papageorgiou 2006). Hinzu kommt der Nachteil, dass sie relativ viele Auswertungen verschiedener Parametersätze benötigen, was aufgrund der hier notwendigen Integration der CTRNN-Gleichungen einen gewissen Rechenaufwand darstellt. Dafür müssen die Sensitivitätsgleichungen, deren Lösung einen Großteil der Rechenzeit in Anspruch nimmt, nicht integriert werden. Ein wichtiger Vorteil der gradientenfreien Lernverfahren in Bezug auf das Training von CTRNN-Netzen ist zudem, dass einige ihrer Vertreter wesentlich bessere Chancen haben, nicht nur ein lokales, sondern auch ein globales Minimum im Vorhersagefehler zu finden. Dies ist besonders bedeutsam im Anbetracht der Tatsache, dass nichtlineare Parameterschätzprobleme, wie sie beim Training von neuronalen Netzen vorliegen, für gewöhnlich eine hohe Anzahl an lokalen Minima aufweisen (Kelley 1999).

MATLAB stellt eine Fülle verschiedener, gradientenfreier Optimierungsverfahren in der *Optimization Toolbox* zur Verfügung, die hierzu genutzt werden können (The MathWorks 2004). Sie können auf einfache Weise aufgerufen werden, sodass an dieser Stelle nicht näher auf eine Implementierung derartiger Verfahren zum Training von CTRNN-Netzen eingegangen wird. Als Referenz sei auf den Anhang A.1 verwiesen, der ein MATLAB-Skript enthält, welches wahlweise ein Mustersuchverfahren, den Simplex-Algorithmus nach NELDER und MEAD oder einen genetischen Algorithmus zum Training des CTRNN heranzieht.

Nach einem Trainingsvorgang ist das ermittelte neuronale Netz stets in seiner Vorhersagequalität zu beurteilen. Hierzu ist es zwingend notwendig, Validierungsdaten bereitzuhalten, die kein Teil der Trainingsdaten waren. Grund hierfür ist, wie in Abschnitt 3.1 bereits ausführlich diskutiert, dass ein neuronales Netz – sofern es genügend Freiheitsgrade besitzt – Trainingsdaten beliebig genau wiedergeben kann, was einer reinen Memorisierung entspricht, nicht aber einer gewünschten Generalisierung. Letztere lässt sich nur durch die Validierung des Netzes mit mindestens einem neuen Datensatz richtig bewerten.

Hierzu ist auch von Bedeutung, anhand welcher Kenngrößen überhaupt die Qualität einer Vorhersage des Ersatzmodells im Vergleich zur realen Systemantwort bewertet wird. Hierzu bieten sich vielerlei Größen an, wie etwa:

- Die größte Abweichung ϵ_∞ zwischen Vorhersage und wahrer Systemantwort. Diese entspricht der Unendlich-Norm der Abweichungen aller Zeitschritte und ist ein physikalisch intuitiv interpretierbares Maß dafür, ob die Vorhersage auf dem ganzen Zeithorizont akzeptabel genau ist, anstatt nur im Mittel eine geringe Abweichung aufzuweisen. Von besonderer Bedeutung ist diese Fehleraussage bei zeitdiskreten NARX-Netzwerken, da hier die eigentliche Dynamik des Ersatzmodells aus der Tatsache folgt, dass die Vorhersage zum aktuellen Zeitschritt direkt aus der der letzten d_y Zeitschritte folgt. Ist nun beispielsweise der Vorhersagefehler über einen gewissen Zeitraum sehr klein, so genügt eine größere Abweichung auf einem engen Zeitraum – gar zu einem einzigen Zeitpunkt – bereits, um die Qualität aller zeitlich folgenden Vorhersagen maßgeblich zu vermindern.

- Die Korrelation der Vorhersage des Ersatzmodells mit der realen Systemantwort. Im Vergleich zur maximalen Abweichung dient die Korrelation als ein Maß für einen allgemeinen ähnlichen Verlauf dieser beiden Zeitsignale und berechnet sich nach (Papula 2008)

$$R^2 = \frac{\mathrm{Cov}^2(y,\hat{y})}{\mathrm{Cov}(y,y) \cdot \mathrm{Cov}(\hat{y},\hat{y})} \quad . \tag{4.1}$$

Eine völlige Übereinstimmung wahren Systemantwort y und ihrer Vorhersage $\hat{y}$ liefert $R^2 = 1$. Die Korrelation ist wesentlich robuster gegenüber einzelnen, starken Abweichungen und stellt somit ein grundsätzlich anderes Bewertungscharakteristikum dar als die maximale Abweichung.

- Die Autokorrelation der Vorhersagefehler (Al Seyab 2006). Dieses Fehlerkriterium entspricht der Forderung, dass die Fehler bei einem guten Ersatzmodell zufällig verteilt sein sollten, anstatt einer Kausalität zu folgen.

- Die Kreuzkorrelation der Fehler mit den Steuerungen $u(t)$ (Al Seyab 2006). Diese sollte niedrig sein, um auszuschließen, dass der Vorhersagefehler eine Abhängigkeit von der Steuerung aufweist.

Für die Bewertung der Güte von Ersatzmodellen wurden im Rahmen dieser Arbeit einerseits die größte Abweichung und andererseits die Korrelation zwischen y und $\hat{y}$ herangezogen. Wie bereits angedeutet, ergänzen sich diese beiden Kennzahlen gegenseitig. Ein qualitativ gutes Ersatzmodell sollte immer einen hohen Korrelationswert bei niedriger maximaler Abweichung aufweisen. Diese Bewertung ist allerdings nicht immer eindeutig. So kann ein Modell einen besseren Korrelationswert und gleichzeitig eine größere maximale Abweichung als ein Anderes haben. Daher wurden bei der Auswertung stets beide Größen berücksichtigt. Ein MATLAB-Skript zur Auswertung und zum visuellen Vergleich mehrerer Netze ist im Anhang A.1 zu finden.

Zum Abschluss dieses Abschnitts sollen noch die individuellen Vor- und Nachteile der zeitdiskreten gegenüber den zeitstetigen neuronalen Netzwerken erläutert werden, um eine Entscheidungshilfe zur Auswahl der Netzart zu geben.

So lässt sich für die zeitdiskreten NARX-Netze, wie sie hier präsentiert wurden, festhalten, dass aufgrund der benutzerfreundlichen *Neural Network*-Toolbox auf angenehme Weise ein einfacher, effizienter und robuster Trainingsvorgang erfolgen kann. Nachteilig wirkt sich allerdings aus, dass gemäß Abschnitt 4.1 eine Umwandlung in ein zeitstetiges Ersatzmodell notwendig ist, wenn eine zeitstetige Optimierung vorgenommen werden soll, wodurch eine weitere Quelle von numerischen Fehlern in das Verfahren aufgenommen werden muss. Hinzu kommt, dass aufgrund des hohen Entwicklungsstands der besagten Toolbox auch eine Vielzahl an Trainingsparametern einstellbar ist, was tendenziell eine Unsicherheit darstellt. So sind natürlich die Netztopologie, aber auch die Aufteilung der gesamten Datensätze in Trainings- und Validierungsdaten, die Wahl des Trainingsverfahrens an sich sowie das dem Training zugrundeliegende Gütekriterium zu wählen, wobei die richtige Wahl dieser Optionen durchaus problemabhängig ist.

Demgegenüber erhält man beim Heranziehen von zeitstetigen CTRNN-Netzen ein Ersatzmodell, das das dynamische Originalsystem in einer natürlichen, differentiellen Form beschreibt. Dies führt insbesondere auch dazu, dass keine Umwandlung notwendig ist, wie dies bei NARX-Netzen der Fall ist. Diesem großen Vorteil stehen jedoch auch einige Nachteile entgegen. So ist ganz generell der Trainingsprozess für CTRNN-Netze wesentlich aufwändiger und rechenzeitintensiver. Darüber hinaus ist es beispielsweise nicht möglich, bei einer vorgegebenen Steuerung $u(t)$ auch deren Ableitung implizit im Ersatzmodell zu berücksichtigen. Während dies bei NARX-Netzen durch das Einbeziehen früherer Werte von $u(t)$ implizit möglich ist, muss hierzu bei CTRNN-Netzen die Ableitung von $u(t)$ selbst dem Netz als weitere Steuerung zugeführt werden. Dies birgt jedoch weitere potentielle Probleme. So erhöht sich dadurch die Dimensionalität des CTRNN-Systems, sodass mehr Freiheitsgrade im Trainingsprozess zu ermitteln sind. Des Weiteren ist a priori eine sorgfältige Aufbereitung der Trainingsdaten notwendig, denn die numerische Differentiation von $u(t)$ vor dem Training kann, je nach Verlauf der Steuerung, zu Ableitungen führen, die ihren Wert auf kurzen Zeitskalen stark ändern. Dies führt unter Umständen zu Schwierigkeiten bei der numerischen Integration der CTRNN-Gleichungen während des Trainingsprozesses, da hierdurch das Differentialgleichungssystem einerseits steifer wird und andererseits aufgrund von variablen Zeitschrittweiten des Integrationsverfahrens wichtige Sprünge oder schnelle Änderungen in der Ableitung möglicherweise ganz übersprungen werden.

Nachdem im vorigen Abschnitt das Training der verschiedenen neuronalen Netzwerke im Mittelpunkt stand, soll nun vertieft auf die mathematische Umwandlung von zeitdiskreten Netzen in ihre zeitstetigen Pendants eingegangen werden. Die Notwendigkeit dieser Umwandlung ergibt sich gemäß den in Abschnitt 4.1 angestellten Überlegungen, wenn eine zeitstetige Optimierung mittels des Kollokationsverfahrens gewünscht wird, jedoch nur ein zeitdiskretes Ersatzmodell vorliegt.

Die bestimmende Gleichung eines NARX-Netzwerks, Gleichung (3.6), beschreibt das physikalische System mithilfe eines zeitdiskreten Ersatzmodells. Um hieraus ein zeitstetiges Ersatzmodell zu gewinnen, wird auf Differenzenquotienten zurückgegriffen. Damit lassen sich beispielsweise zwei zeitlich aufeinanderfolgende Werte von $\hat{y}$ unter Berücksichtigung einer Zeitschrittweite $\triangle t$ approximierend in ein Differential erster Ordnung überführen[1]. Je mehr zeitlich aufeinanderfolgende, diskrete Werte von $\hat{y}$ man berücksichtigt, desto höher ergibt sich die Ordnung des entsprechenden Differentials. Führt man dies beispielhaft für ein neuronales Netz mit $d_u = d_y = 2$ durch, ergibt sich:

$$\hat{y}^{k-1} = y_{\mathrm{EM}} - \dot{y}_{\mathrm{EM}} \cdot \triangle t \tag{4.2a}$$
$$\hat{y}^{k-2} = y_{\mathrm{EM}} - 2\dot{y}_{\mathrm{EM}} \cdot \triangle t + \ddot{y}_{\mathrm{EM}} \cdot \triangle t^2 \tag{4.2b}$$
$$u^{k-1} = u^k - \dot{u} \cdot \triangle t \tag{4.2c}$$
$$u^{k-2} = u^k - 2\dot{u} \cdot \triangle t + \ddot{u} \cdot \triangle t^2 \tag{4.2d}$$

Mit dem Index EM sind hier die Größen des stetigen Ersatzmodells bezeichnet. Im Allgemeinen Fall kann man die entsprechenden zeitkontinuierlichen Pendants wie folgt bilden:

$$\hat{y}^{k-i} = \sum_{j=0}^{i} \gamma_{i,j} \triangle t^j \cdot \frac{\mathrm{d}^j}{\mathrm{d}t^j} y_{\mathrm{EM}} \tag{4.3a}$$
$$u^{k-i} = \sum_{j=0}^{i} \gamma_{i,j} \triangle t^j \cdot \frac{\mathrm{d}^j}{\mathrm{d}t^j} u \tag{4.3b}$$

Der j-te Koeffizient $\gamma_{i,j}$ für die i-mal verzögerte Eingangsgröße ist dabei aus Tabelle 4.1 ablesbar.

	$j=0$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$
$i=0$	1	0	0	0	0	0
$i=1$	1	-1	0	0	0	0
$i=2$	1	-2	1	0	0	0
$i=3$	1	-3	3	-1	0	0
$i=4$	1	-4	6	-4	1	0
$i=5$	1	-5	10	-10	5	-1

Tabelle 4.1.: Die Koeffizienten $\gamma_{i,j}$ zur Berechnung von vergangenen Werten von $\hat{y}$ und u aus ihren Ableitungen gemäß den Gleichungen (4.3a) und (4.3b).

Mit den Gleichungen (4.3a) und (4.3b) lässt sich nun, ausgehend von Gleichung (3.6), durch Substitution eine stetige Formulierung für den Ausgang des i-ten Neurons der versteckten Schicht, mit H_i bezeichnet, gewinnen. So gilt zunächst wiederum im Sonderfall $d_u = d_y = 2$:

$$H_i = \begin{cases} 1 & i = 0 \\ \tanh\left[\kappa_i + \beta_{i,1} \cdot u + \beta_{i,2} \cdot \dot{u} + \beta_{i,3} \cdot \ddot{u} + \delta_{i,1} \cdot y_{\mathrm{EM}} + \delta_{i,2} \cdot \dot{y}_{\mathrm{EM}} + \delta_{i,3} \cdot \ddot{y}_{\mathrm{EM}}\right] & \text{sonst} \end{cases} \tag{4.4}$$

[1] für u gilt die Betrachtung vollständig analog.

Für allgemeine Fälle gilt in Analogie:

$$H_i = \begin{cases} 1 & i = 0 \\ \tanh\left[\kappa_i + \sum_{k=0}^{d_u} \beta_{i,k+1} \cdot \frac{\mathrm{d}^k}{\mathrm{d}t^k} u + \sum_{j=0}^{d_y} \delta_{i,j+1} \cdot \frac{\mathrm{d}^j}{\mathrm{d}t^j} y_{\mathrm{EM}}\right] & \text{sonst} \end{cases} \tag{4.5}$$

Der Sonderfall für $i = 0$ wurde hier nur aus Gründen der nachfolgenden Notation eingeführt. Die Zusammensetzung dieses Ausdrucks lässt sich leicht anhand der Abbildung 3.2 nachvollziehen, die den allgemeinen Aufbau eines NARX-Netzwerks zeigt. Die in der Gleichungen (4.5) eingeführten Konstanten β, δ und κ sind eindeutig durch die Gewichte des trainierten, diskreten neuronalen Netzes sowie $\triangle t$ bestimmt. So gilt für den Sonderfall $d_u = d_y = 2$:

$$\beta_{i,1} = w_{IH,1,i} + w_{IH,2,i} \tag{4.6a}$$
$$\beta_{i,2} = -\triangle t \cdot w_{IH,1,i} - 2\triangle t \cdot w_{IH,2,i} \tag{4.6b}$$
$$\beta_{i,3} = \triangle t^2 \cdot w_{IH,2,i} \tag{4.6c}$$
$$\delta_{i,1} = w_{IH,3,i} + w_{IH,4,i} \tag{4.6d}$$
$$\delta_{i,2} = -\triangle t \cdot w_{IH,3,i} - 2\triangle t \cdot w_{IH,4,i} \tag{4.6e}$$
$$\delta_{i,3} = \triangle t^2 \cdot w_{IH,4,i} \tag{4.6f}$$
$$\kappa_i = b_{H,i} \tag{4.6g}$$

Für allgemeine Fälle bis zur Verzögerungsordnung 5 gelten die Formulierungen gemäß Tabelle 4.2.

Weiter ergibt sich nach Gleichung (3.6) der gesamte Output des neuronalen Netzes, beziehungsweise seines stetigen Substituts y_{EM}, zu:

$$y_{\mathrm{EM}} = \sum_{i=0}^{h} \alpha_i H_i \tag{4.7}$$

Die Konstanten α_i ergeben sich auf einfache Weise aus der Topologie des neuralen Netzes und werden durch den Übergang auf das stetige Ersatzmodell nicht beeinflusst. Daher gilt, wie man sich anhand Abbildung 3.2 leicht vergewissert, im Allgemeinen:

$$\alpha_0 = b_O \tag{4.8}$$
$$\alpha_i = w_{HO,i} \tag{4.9}$$

Im Anhang A.1 befindet sich ein MATLAB-Skript, das ein NARX-Netzwerk gemäß der vorstehend beschriebenen Berechnungsmethode in ein System von zeitstetigen Differenzialgleichungen überführt.

4.5 Lösen des Optimalsteuerungsproblems

Wurde nun in der ersten Phase des Gesamtverfahrens ein qualitativ gutes Ersatzmodell erzeugt, kann in der zweiten Phase die eigentliche Optimierung vonstatten gehen. Hierzu muss wiederum unterschieden werden, ob ein zeitdiskretes oder zeitstetiges neuronales Netz als Ersatzmodell vorliegt.

Zunächst wird das weitere Vorgehen anhand der zeitstetigen Variante erläutert. Das Ersatzmodell, das entweder als CTRNN-Netz oder eine in ein zeitstetiges System umgewandelte Variante eines NARX-Netzes vorliegt, stellt somit die ehemals unbekannte Systemfunktion $f(x,u,t)$ in Gleichung (3.22b) des Optimalsteuerungsproblems dar. Da diese nun vorliegt, steht prinzipiell einer Lösung dieses Problems nichts mehr im Wege. Wie bereits in Abschnitt 3.2.4 dargelegt, soll im Rahmen dieser Arbeit die frei erhältliche Open-Source-Software JMODELICA.ORG herangezogen werden (Akesson u. a. 2009). Diese baut

	$\beta_{i,1}$	$\beta_{i,2}/\triangle t$	$\beta_{i,3}/\triangle t^2$
$d_u = 1$	$w_{1,i}$	$-w_{1,i}$	
$d_u = 2$	$w_{1,i} + w_{2,i}$	$-w_{1,i} - 2w_{2,i}$	$w_{2,i}$
$d_u = 3$	$w_{1,i} + w_{2,i} + w_{3,i}$	$-w_{1,i} - 2w_{2,i} - 3w_{3,i}$	$w_{2,i} + 3w_{3,i}$
$d_u = 4$	$w_{1,i} + w_{2,i} + w_{3,i} + w_{4,i}$	$-w_{1,i} - 2w_{2,i} - 3w_{3,i} - 4w_{4,i}$	$w_{2,i} + 3w_{3,i} + 6w_{4,i}$
$d_u = 5$	$w_{1,i} + w_{2,i} + w_{3,i} + w_{4,i} + w_{5,i}$	$-w_{1,i} - 2w_{2,i} - 3w_{3,i} - 4w_{4,i} - 5w_{5,i}$	$w_{2,i} + 3w_{3,i} + 6w_{4,i} + 10w_{5,i}$

	$\delta_{i,4}/\triangle t^3$	$\delta_{i,5}/\triangle t^4$	$\delta_{i,6}/\triangle t^5$
$d_u = 1$			
$d_u = 2$			
$d_u = 3$	$-w_{3,i}$		
$d_u = 4$	$-w_{3,i} - 4w_{4,i}$	$w_{4,i}$	
$d_u = 5$	$-w_{3,i} - 4w_{4,i} - 10w_{5,i}$	$w_{4,i} + 5w_{5,i}$	$-w_{5,i}$

	$\delta_{i,1}$	$\delta_{i,2}/\triangle t$	$\delta_{i,3}/\triangle t^2$
$d_y = 1$	$w_{6,i}$	$-w_{6,i}$	
$d_y = 2$	$w_{6,i} + w_{7,i}$	$-w_{6,i} - 2w_{7,i}$	$w_{7,i}$
$d_y = 3$	$w_{6,i} + w_{7,i} + w_{8,i}$	$-w_{6,i} - 2w_{7,i} - 3w_{8,i}$	$w_{7,i} + 3w_{8,i}$
$d_y = 4$	$w_{6,i} + w_{7,i} + w_{8,i} + w_{9,i}$	$-w_{6,i} - 2w_{7,i} - 3w_{8,i} - 4w_{9,i}$	$w_{7,i} + 3w_{8,i} + 6w_{9,i}$
$d_y = 5$	$w_{6,i} + w_{7,i} + w_{8,i} + w_{9,i} + w_{10,i}$	$-w_{6,i} - 2w_{7,i} - 3w_{8,i} - 4w_{9,i} - 5w_{10,i}$	$w_{7,i} + 3w_{8,i} + 6w_{9,i} + 10w_{10,i}$

	$\delta_{i,4}/\triangle t^3$	$\delta_{i,5}/\triangle t^4$	$\delta_{i,6}/\triangle t^5$
$d_y = 1$			
$d_y = 2$			
$d_y = 3$	$-w_{8,i}$		
$d_y = 4$	$-w_{8,i} - 4w_{9,i}$	$w_{9,i}$	
$d_y = 5$	$-w_{8,i} - 4w_{9,i} - 10w_{10,i}$	$w_{9,i} + 5w_{10,i}$	$-w_{10,i}$

Tabelle 4.2.: Berechnungsvorschriften für die Koeffizienten β und δ. Der Übersicht halber steht w abkürzend für w_{IH}.

auf der bekannten Modellierungssprache MODELICA auf, wobei sie eine Spracherweiterung namens OPTIMICA einführt. Diese ermöglicht die Spezifikation von Optimalsteuerungsproblemen, die der Form der Gleichungen (3.22a) bis (3.22c) entsprechen, wobei deren Formulierung auf einfache Weise gleichungsbasiert vorgenommen werden kann. Weiterhin erlaubt sie die flexible Angabe von Beschränkungen aller Art[2] und verschiedenen Kostenfunktionalen (Akesson 2008). Das über OPTIMICA spezifizierte Optimalsteuerungsproblem wird dann innerhalb von JMODELICA.ORG mittels eines direkten Kollokationsverfahrens, wie es bereits in Abschnitt 3.2.4 eingeführt wurde, in ein endlich-dimensionals NLP transformiert. Dieses wird über eine Schnittstelle an die ebenfalls frei erhältliche Software IPOPT übergeben, die derartige Probleme mit einem innere-Punkte-Verfahren löst und die Ergebnisse zurück an JMODELICA.ORG liefert. Dieser Prozess wird im Rahmen einer Iteration einige Male wiederholt, bis das Kollokationsverfahren gegen die Lösung des Optimalsteuerungsproblems konvergiert (Magnusson 2012, Modelon AB 2014).

Ein Optimica-Skript zur Spezifikation eines Optimalsteuerungsproblems besteht dabei aus

- dem Kopf des Skriptes mit Definition der Kostenfunktion sowie der Start- und der Endzeit (falls diese nicht frei ist),
- Deklarationen aller Variablen und (mindestens) einer Steuerungsvariable,
- einem Equation-Block, der die Modellgleichungen des zu optimierenden Systems beinhaltet, sowie
- einem Constraint-Block, der Endbedingungen und Beschränkungen enthält.

Ein diesem Aufbau entsprechendes Skript ist der Arbeit in Anhang A.2 beigefügt. Das Aufrufen des Lösers erfolgt letztlich über ein Python-Kommandozeilenprogramm, wobei dem Löser gewisse Parameter übergeben werden können. Wichtige Parameter sind beispielsweise:

- Die Anzahl der Kollokationselemente, in die das gesamte Zeitintervall aufgeteilt wird.
- Der Grad der Kollokationspolynome in jedem Segment.
- Die Diskretisierungsmethode, beispielsweise Gauß- oder Lobatto-Kollokation.
- Die Spezifikation einer Startschätzung für die optimale Steuerung. Dieses Option kann von zentraler Bedeutung sein, wenn die Optimierungsprobleme schwer lösbar sind. In diesem Fall konvergiert das Optimierungsverfahren oftmals nur, wenn man das Problem zunächst nur mit schwachen Beschränkungen, Toleranzen und wenig Elementen löst und dessen Lösung als Startschätzung für zunehmend stärker restringierte Probleme mit mehr Elementen verwendet (Modelon AB 2014).
- Verschiedene Toleranzen, beispielsweise für die NLP-Lösung in IPOPT.
- Maximale Anzahl von Iterationen in IPOPT.

Wie diese Optionen spezifizierbar sind, sei an dieser Stelle nicht eingehend behandelt, sondern stattdessen auf die Dokumentation von JMODELICA.ORG verwiesen (Modelon AB 2014). Ein typisches Python-Skript, das diese und andere Optionen festlegt, den Löser aufruft sowie die Optimierungsergebnisse visualisiert, ist in Anhang A.2 zu finden.

Während die vorherigen Betrachtungen für die zeitstetige Optimierung mit zeitstetigen Ersatzmodellen (*CNCO* gemäß Abschnitt 4.1) galten, soll im Folgenden der Fokus auf die zeitdiskrete Optimierung mit einem zeitdiskreten NARX-Modell (*DNDO*) gelegt werden. Hierzu bietet sich das direkte Schießverfahren an, das bereits in Abschnitt 3.2.4 kurz thematisiert wurde. Dieses wurde im Rahmen dieser Arbeit in MATLAB mit einem SQP-Verfahren implementiert. In jeder SQP-Iteration wird die aktuelle

[2] Hiervon ausgenommen sind innere-Punkte-Beschränkungen, die jedoch in praxisnahen Optimierungsproblemen eher selten auftreten (Modelon AB 2014)

suboptimale Steuerung in das NARX-Netzwerk eingespeist und die Systemantwort in Hinblick auf die Kostenfunktion und Verletzung der Nebenbedingungen ausgewertet. Diese Informationen nutzt dann der SQP-Algorithmus, um eine neue, bessere Steuerung zu generieren. Eine Implementierung dieses Vorgehens befindet sich im Anhang A.2.

Das *DNDO*-Verfahren bietet den Vorteil, dass es direkt und ohne weitere Zwischenschritte auf ein NARX-Netz anwendbar ist. Dieses kann zudem beliebig komplex sein, ohne das grundlegende Vorgehen zu beeinflussen. Eine Umwandlung hingegen wird gemäß Abschnitt 4.4 immer umso aufwändiger, je komplexer das NARX-Netz ist. Darüber hinaus enthält das entstehende stetige Modell bei höheren Verzögerungen auch Ableitungen immer höherer Ordnung, die die Lösbarkeit des zu optimierende Systems tendenziell erschweren. Diese Probleme können mit der *DNDO*-Vorgehensweise umgangen werden.

4.6 Fehlerbetrachtung

Zum Abschluss der Verfahrensentwicklung soll in diesem Abschnitt eine Betrachtung der Fehler erfolgen, die im Laufe der verschiedenen Verfahrensschritte gemacht werden.

Zunächst muss beachtet werden, dass bereits in der Erzeugung der Trainingsdaten Fehler gemacht werden können, die einen Einfluss auf die Qualität des Endergebnisses haben. Dies ist beispielsweise der Fall, wenn die Trainingsdaten aus einem Simulationsmodell bezogen werden, welches jedoch in aller Regel die Realität nicht in bis ins letzte Details wiedergeben kann. Auf ähnliche Weise ist zu erwarten, dass Trainingsdaten, die aus einem experimentellen Aufbau bezogen werden, ebenso eine gewisse Verfälschung gegenüber der Realität aufweisen. Aus diesen Gründen sollte auf die Qualität der Trainingsdaten ein verstärktes Augenmerk gelegt werden, wie in Abschnitt 3.1.5 bereits erläutert wurde.

Als nächste Fehlerquelle ist die Substitution des Originalsystems durch ein Ersatzmodell zu nennen. Auf Basis von Erfahrungen sowie der in den letzten Kapiteln gemachten Aussagen kann davon ausgegangen werden, dass der bei diesem Schritt induzierte Fehler die größte Fehlerquelle im gesamten Verfahren sein dürfte. Aus diesem Grund ist beim Erzeugen des Ersatzmodells ein besonderes Augenmerk darauf zu legen, ein möglichst gut generalisiertes neuronales Netz mit geringen Vorhersagefehlern zu erreichen.

Auch durch das Ableiten eines zeitstetigen Ersatzmodells aus einem zeitdiskreten NARX-Netz werden Approximationsfehler gemacht. Diese sind umso größer, je größer die Zeitschrittweite $\triangle t$ ist. Daher sollte diese unbedingt so früh wie möglich und sorgfältig festgelegt werden, da diese auch für die Erzeugung der Trainingsdaten von Bedeutung ist.

Eine weitere Fehlerquelle liegt auch in der numerischen Lösung des Optimalsteuerungsproblems. Die dort gemachten Fehler sind von vielen Faktoren abhängig, wie etwa der Anzahl der Elemente beim Kollokationsverfahren beziehungsweise der Anzahl der Stützstellen der Steuerung im Falle des Schießverfahrens. Diese Fehler sind jedoch gut kontrollierbar. Einerseits hat man über die zu wählenden Toleranzen und Segmentierung des Zeitintervalls relativ weitreichende Mittel zur Steuerung der Genauigkeit. Andererseits lässt sich die Genauigkeit der Lösung des Optimalsteuerungsproblems jederzeit überprüfen, indem die erhaltene optimale Steuerung in das Ersatzmodell eingespeist wird und dessen Systemantwort mit dem Optimierungsergebnis verglichen wird. Weichen die beiden Trajektorien nennenswert voneinander ab, sollten die Zeitintervalle enger gewählt und beim Kollokationsverfahren gegebenenfalls Ansatzpolynome höherer Ordnung eingesetzt werden.

Letztlich soll noch angeschnitten werden, dass selbst ein sehr genaues Ergebnis für die optimale Steuerung noch kein Garant für eine wirklich optimale Gestaltung des realen Prozesses ist. Dies liegt vornehmlich daran, dass die optimale Steuerung nur für den spezifizierten Anfangszustand gültig ist. Das hat zur Folge, dass die Optimalität der Prozessführung unmittelbar verloren geht, wenn dieser Störungen unterliegt oder unter geänderten Bedingungen betrieben wird. Aus diesem Grund wäre es für zukünftige Arbeiten sinnvoll, von einer optimalen Steuerung auf eine optimale Regelung überzugehen, die stetig unter Beachtung der tatsächlichen Zustände des Prozesses eine neue optimale Steuerung einspeist. Die Ermittlung derartiger optimaler Regelungen ist leider aufwändig und mit sehr großen Schwierigkeiten verbunden (Papageorgiou 2006, Kirk 1970). Es gibt allerdings Ansätze zur numerischen Synthese opti-

maler Regelungen aus vielen, einzelnen optimalen Steuerungen. Derartige Methoden werden Verfahren der benachbarten Extremalen genannt. Für deren detaillierte Abhandlung sei auf die Spezialliteratur verweisen (Milam 2003, Bourdache-Siguerdidjane u. Fliess 1987, Hemami u. a. 1992, Ross u. a. 2008). Eine weitere, weitaus einfachere Methode ist eine einfache Regelung der realen Zustände des Prozesses an die optimale Zustandstrajektorien aus dem Optimierungsergebnis heran. Mit dieser Sollwerttrajektorien-Folgeregelung geht allerdings ebenfalls die Optimalität der Steuerung verloren. Darüber hinaus können Beschränkungen durch den Regeleingriff verletzt werden (Milam 2003).

5 Zwischenfazit

Mit den gemachten Überlegungen ergibt sich also ein Gesamtverfahren im Sinne eines Vorgehens, das verschiedene Methoden der Systemidentifkation und der Optimierung von Steuerung miteinander verkoppelt. Je nach Kombination verschiedener Methoden wurden drei Vorgehenspfade unterschieden, die entsprechend ihres diskreten oder stetigen Charakters bei der Systemidentifikation oder der Optimierung mit DNDO, DNCO oder CNCO bezeichnet wurden. Jede dieser Pfade hat seine eigenen Vorteile, Nachteile, Notwendigkeiten und ggf. auch Fehlerquellen, die in Kapitel 4 ausführlich dargelegt wurden.

An dieser Stelle soll an die formulierten Anforderungen aus Abschnitt 2.1 zurückgeblickt werden, um das Gesamtverfahren hinsichtlich seiner Qualität zu beurteilen. So kann festgehalten werden, dass das Verfahren unabhängig vom physikalischen System anwendbar ist, da neuronale Netzwerke aufgrund ihrer Eigenschaft als universaler Approximator prinzipiell für die Repräsentation jedweder funktionaler Zusammenhänge geeignet sind. Somit weist das Verfahren eine hohe Allgemeinheit gegenüber den Eingangsdaten und der zu untersuchenden Zusammenhänge auf. Darüber hinaus ist es möglich, verschiedenste Kostenfunktionen und Nebenbedingungen in die Lösung des Optimalsteuerungsproblems einfließen zu lassen, sodass auch die Forderung der Flexibilität erfüllt werden konnte. Die erreichbare Genauigkeit des Verfahrens ist stark abhängig von der Qualität des Ersatzmodells, welche wiederum problemabhängig ist. Prinzipiell sind mit neuronalen Netzwerken hohe Genauigkeiten erreichbar, im Einzelfall könnte der Trainingsprozess jedoch schwierig und zeitaufwändig werden. Von Seiten der Effizienz und Robustheit, die ebenfalls gefordert wurden, ist festzuhalten, dass die Trainingsverfahren für die neuronalen Netze im Rahmen ihrer stochastischen Komponente, den Anfangswerten, zuverlässig sind. Dieser Zufallseinfluss ist jedoch durch mehrere Wiederholungen des Trainings kontrollierbar. Nicht zuletzt aus diesem Grund verlangt der Schritt Systemidentifikation auch den größten Aufwandsanteil des Gesamtverfahrens. Dies wird durch die Notwendigkeit verstärkt, verschiedene Netzkomplexitäten zu beurteilen, um eine geeignete Generalisierung der neuronalen Netze zu erreichen. Alles in allem wurde also ein Verfahren entwickelt, das prinzipiell die Anforderungen im Großen und Ganzen erfüllt. Die Performanz und die Ergebnisqualität des Verfahrens ist jedoch problemabhängig zu beurteilen.

Die Implementierung der Systemidentifikation wurde vornehmlich in MATLAB vorgenommen, wobei von bestehenden Funktionsbibliotheken Gebrauch gemacht wurde. Zur Optimierung wurde je nach gewählter Methode entweder ebenso MATLAB oder die freie Software JMODELICA.ORG herangezogen. Ergebnis dieser Arbeit sind fertige MATLAB-, Python- und OPTIMICA-Skripte, die vom Benutzer lediglich an seinen Anwendungsfall anzupassen sind.

Das fertige Gesamtverfahren soll im nun folgenden Teil II der vorliegenden Arbeit anhand einiger Beispielprobleme in seiner Anwendung demonstriert werden.

Teil II.
Benchmarking der Verfahren

6 Optimales Umschalten einer Mischungsstrecke

6.1 Problembeschreibung und Modellbildung

Als erstes, grundlegendes Benchmark-Problem zum Erproben des im ersten Teil dieser Arbeit entwickelten Optimierungsverfahrens soll eine einfache Mischungsstrecke nach Abbildung 6.1 betrachtet werden. Diese wurde mit der Modellierungssprache MODELICA in der Simulationsumgebung DYMOLA abgebildet. Die Aufgabe der Mischungsstrecke ist es, einen heißen und einen kalten Luftstrom in variablen Verhältnissen zu vermischen und nach der Vermischung durch einen dickwandigen Strömungskanal zu leiten. Dabei beträgt die Temperatur der heißen Luftquelle $T_{\text{kalt}} = 50\,°\text{C}$, die der heißen Quelle $T_{\text{heiß}} = 200\,°\text{C}$. Das Mischungsverhältnis wird durch die Steuerung $u(t)$ bestimmt, die auf jeweils ein Ventil mit linearer Kennlinie hinter der heißen und der kalten Quelle einwirkt. Ist $u = 0\,\%$, so ist die Temperatur nach der Mischung gerade T_{kalt}, für $u = 100\,\%$ ist sie gleich T_{warm}.

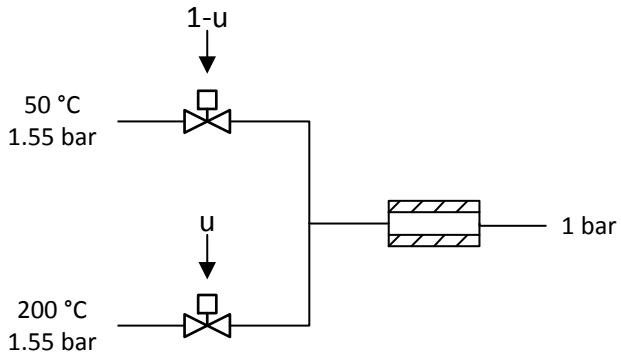


Abbildung 6.1.: Fließdiagramm einer einfachen Mischungsstrecke.

In der Stahlmasse des dickwandigen Strömungskanals entstehen bei der zeitlichen Änderung der Mischungstemperatur thermische Spannungen σ. Die Wand, die einen Innendurchmesser von 0.2 m und eine Dicke von 50 mm hat, wurde in radialer Richtung in 15 Kontrollvolumen diskretisiert, um eine Berechnung der thermischen Spannungen zu ermöglichen. Auf der Außenseite wurde die Wand als adiabat modelliert.

Anhand dieses relativ einfachen Beispiels soll das in Teil I dieser Arbeit entwickelte Verfahren auf folgendes Optimierungsproblem angewendet werden. Zu Beginn betrage u stationär $0\,\%$ und die Temperatur in der Metallmasse sei homogen 50 °C, sodass keine thermischen Spannungen auftreten. Ausgehend von diesem Anfangszustand soll nun so schnell wie möglich die Mischungstemperatur auf $T_{\text{heiß}}$, also u auf $100\,\%$ erhöht werden, ohne dass dabei höhere thermische Spannungen als 150 MPa in der Metallmasse des Strömungskanals induziert werden. Zusätzliche Beschränkungen werden durch die Ventildynamik eingeführt, die eine maximale Stellgeschwindigkeit von $\dot{u}_{\max} = 3\,\%/\text{s}$ und eine maximale Ventilbeschleunigung von $|\ddot{u}|_{\max} = 0.25\,\%/\text{s}^2$ festlegt. Die mathematische Formulierung dieses Optimierungsproblem lautet damit wie folgt:

$$\min_{u \in U} \quad J[u(t)] = t_f \tag{6.1a}$$

$$\text{u.B.v.} \quad \dot{x}(t) = f(x(t), u(t), t) \tag{6.1b}$$

$$u \geq 0\,\% \tag{6.1c}$$

$$u \leq 100\,\% \tag{6.1d}$$

$$\dot{u} \leq 3\,\%/\mathrm{s} \tag{6.1e}$$

$$|\ddot{u}| \leq 0.25\,\%/\mathrm{s}^2 \tag{6.1f}$$

$$|\sigma| \leq 150\,\mathrm{MPa} \tag{6.1g}$$

Als Anfangsbedingungen und Endbedingungen gilt den obigen Ausführungen entsprechend:

$$u(0) = 0\,\% \qquad u(t_f) = 100\,\% \tag{6.1h}$$

$$\dot{u}(0) = 0\,\%/\mathrm{s} \qquad \dot{u}(t_f) = 0 \quad \%/\mathrm{s} \tag{6.1i}$$

$$\ddot{u}(0) = 0\,\%/\mathrm{s}^2 \qquad \ddot{u}(t_f) = 0 \quad \%/\mathrm{s}^2 \tag{6.1j}$$

$$\sigma(0) = 0\,\mathrm{MPa} \tag{6.1k}$$

Um dieses Problem zu lösen, ist es zunächst nötig, die Systemfunktion f, die an dieser Stelle unbekannt sei, entsprechend den Ausführungen im Kapitel 4 durch ein Ersatzmodell $\hat{f}$ zu approximieren. Hierauf wird im folgenden Abschnitt 6.2 eingegangen. Die Lösung des Optimalsteuerungsproblems (6.1a)–(6.1k) steht in Abschnitt 6.3 im Mittelpunkt.

6.2 Bilden eines mathematischen Ersatzmodells

Ausgehend von dem in Dymola erstellen Simulationsmodell der Mischungsstrecke sollen in diesem Abschnitt verschiedene Typen von Ersatzmodellen auf ihre Eignung untersucht werden. Hierzu werden zunächst Trainingsdaten vom Originalmodell abgeleitet und dann verschiedene Ersatzmodelle erzeugt.

6.2.1 Trainingsdatensatz

Wie bereits mehrfach erwähnt, ist die Qualität des Trainingsdatensatzes von hoher Wichtigkeit für die Bildung eines Ersatzmodells. Da in diesem Beispiel ein relativ einfaches Problem mit nur einer Steuerung u und einem Systemausgang y vorliegt, kann auch die notwendige Menge der Trainingsdaten recht klein gehalten werden.

Der Datensatz, der zum Training der neuronalen Netzwerke verwendet wurde, ist in Abbildung 6.2 dargestellt. Validierungsdaten zur Überprüfung der Qualität des Ersatzmodells zeigt Abbildung 6.3. Die Trainingsdaten enthalten einerseits viele schnellstmögliche Ventilöffnungen und -schließungen auf verschiedene Öffnungsgrade und andererseits langsamere, lineare Verläufe der Ventilposition. Die Trainingsdaten haben eine Zeitschrittweite von einer Sekunde.

6.2.2 Physikalisch motiviertes Ersatzmodell

Wie in Kapitel 2 erwähnt wurde, kann bei einfachen Problemen, über die physikalische Hintergrundkenntnis vorliegt, ein Ersatzmodell gebildet werden, welches auf naturwissenschaftlichen Grundprinzipien beruht. Auf diese Weise lässt sich auch für das hier vorliegende System feststellen, dass die Eingangstemperatur in den dickwandigen Behälter in erster Näherung durch eine lineare Interpolation zwischen den Temperaturen T_{warm} und T_{kalt} gegeben ist:

$$T = \xi_1 \cdot u + \xi_2 \tag{6.2}$$

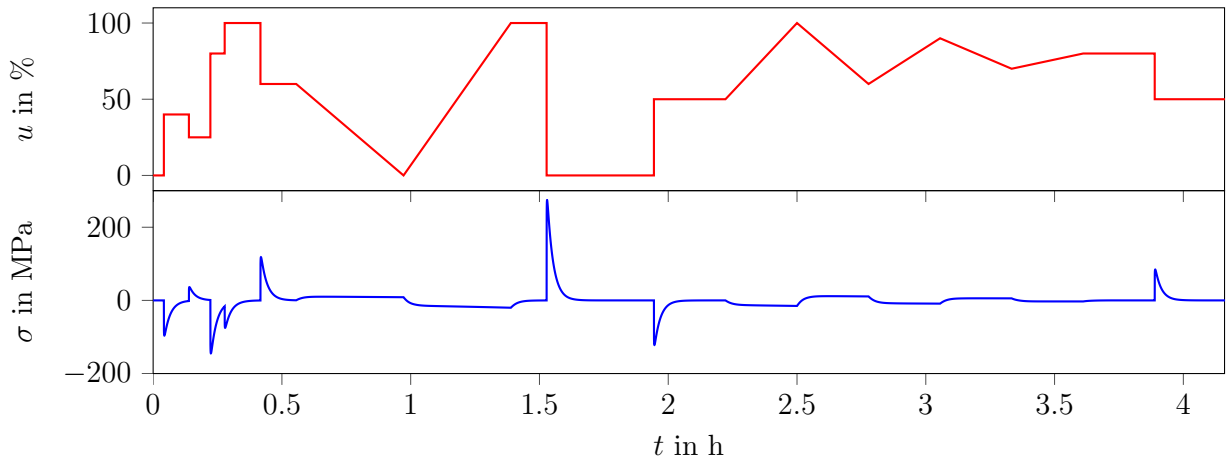


Abbildung 6.2.: Trainingsdaten zur Bildung eines Ersatzmodells der Mischungsstrecke.

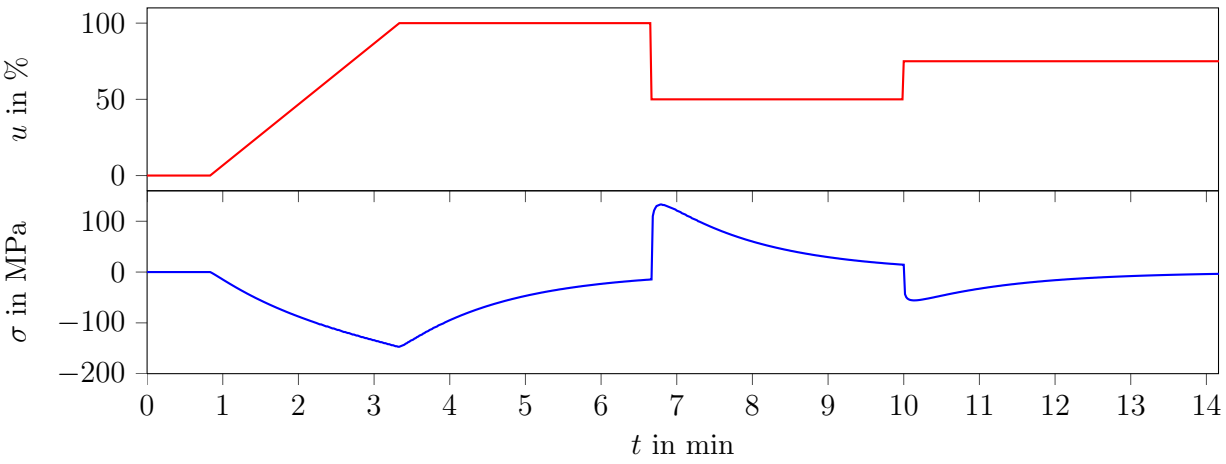


Abbildung 6.3.: Validierungsdaten zur Bildung eines Ersatzmodells der Mischungsstrecke.

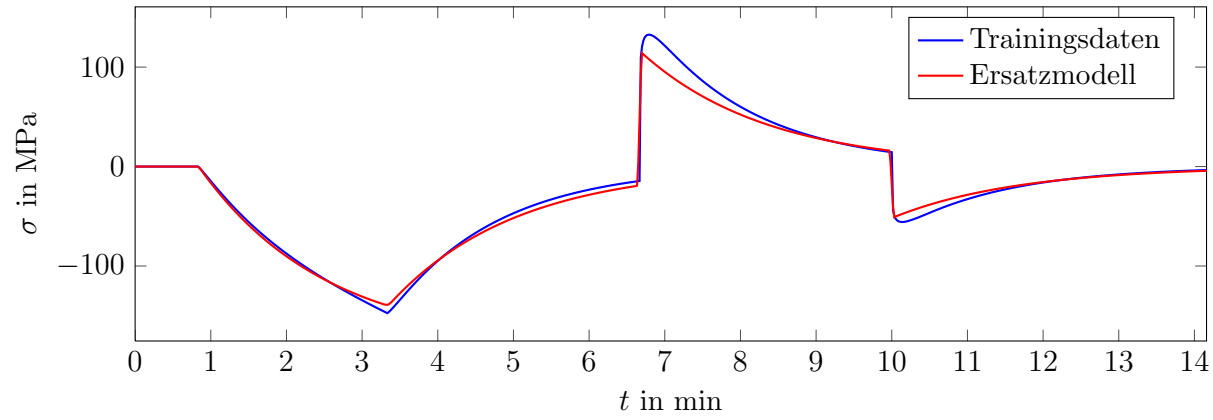


Abbildung 6.4.: Vorhersagequalität des physikalisch motivierten Ersatzmodells.

Hierin sind ξ_1 und ξ_2 Modellparameter. Die Annahme der linearen Interpolation ist näherungsweise erfüllt, wenn die beiden vermischten Einzelströme ähnlich große Wärmekapazitäten $c_{p,\mathrm{warm}}\left(T_{\mathrm{warm}}\right)$ beziehungsweise $c_{p,\mathrm{kalt}}\left(T_{\mathrm{kalt}}\right)$ aufweisen (aufgrund der unterschiedlichen Temperaturen müssen diese unterschiedlich sein) und zudem die Massenströme $\dot{m}_{\mathrm{kalt}}$ und $\dot{m}_{\mathrm{warm}}$ etwa gleich groß sind. Letzteres kann wegen $p_{\mathrm{kalt}} = p_{\mathrm{warm}}$ näherungsweise als erfüllt angesehen werden, da der Dichteunterschied der beiden Einzelströme infolge unterschiedlicher Temperaturen hier eine untergeordnete Rolle spielt.

Für die axialen thermischen Spannungen gilt an der Innenseite von radialsymmetrischen Hohlkörpern die folgende Proportionalität (Epple u. a. 2012):

$$\sigma = \sigma_{\mathrm{th},i} \sim \left(T_{\mathrm{Wand}}\left(r_{\mathrm{i}}\right) - \frac{2}{r_{\mathrm{a}}^2 - r_{\mathrm{i}}^2} \int_{r_{\mathrm{in}}}^{r_{\mathrm{a}}} T_{\mathrm{Wand}}(r) \cdot r \,\mathrm{d}r \right) \tag{6.3}$$

Auf Basis dieser Beobachtung lässt sich nun ein einfaches Modell postulieren. So ist einerseits festzustellen, dass der Betrag und die Änderung der thermischen Spannungen nicht explizit abhängig von der Temperatur des Mediums ist, sondern in letzter Konsequenz abhängig von der Änderungsrate dieser Temperatur. Darüber hinaus ist im stationären Zustand keine Änderung in den Spannungen mehr gegeben. Andererseits ist wohlbekannt, dass thermische Spannungen immer die Tendenz haben, sich abzubauen. Setzt man diesen Abbauprozess als exponentiell abklingend an und bezieht die erstgenannten Beobachtungen mit ein, erhält man das folgende lineare Modell für σ:

$$\dot{\sigma} = \xi_3 \cdot \dot{T} - \xi_4 \cdot \sigma \tag{6.4}$$

Kombiniert man nun die Gleichungen (6.2) und (6.4) zu einem Gesamtmodell, erhält man:

$$\dot{\sigma} = \xi_5 \cdot \dot{u} - \xi_4 \cdot \sigma \ . \tag{6.5}$$

Die zwei Modellparameter ξ_4 und ξ_5 können nun mithilfe üblicher residuenminimierender oder mit allgemeinen statischen Optimierungsalgorithmen bestmöglich ermittelt werden. So konnten die Parameter konnten mithilfe des Simplex-Algorithmus nach Nelder und Mead auf einfache Weise zu $\xi_4 \approx 0.01$ und $\xi_5 \approx -0.27$ bestimmt werden (Nelder u. Mead 1965). Die Vorhersage des Ersatzmodells für die Validierungsdaten sind in Abbildung 6.4 dargestellt. Es ist deutlich erkennbar, dass das simple lineare Ersatzmodell bei dem hier vorliegenden, recht einfachen Problem bereits relativ gute Ergebnisse liefert. In der Tat sind die Vorhersage des Ersatzmodells und die Validierungsdaten zu 99.4% korreliert. Bei näherem Hinsehen fällt jedoch auf, dass die Vorhersage des Ersatzmodells sich direkt nach Sprüngen der Ventilstellungen u trotz des guten Korrelationswertes qualitativ vom realen Verlauf unterscheidet. Offenbar ist das lineare Ersatzmodell nicht in der Lage, die Trägheitseigenschaft der thermischen Spannungen abzubilden: Nach einem abgeschlossenen Sprung in u steigen die Spannungen noch eine gewisse Zeit weiter, was das Ersatzmodell aufgrund seiner Linearität, Ordnung und alleiniger Abhängigkeit von $\dot{u}$ nicht wiedergeben kann. Dass eine lineare Modellbildung nicht ausreichend ist, lässt sich auch in Gleichung (6.3) ersehen: Hier wird ersichtlich, dass die thermischen Spannungen von der Differenz zwischen der mittleren Wandtemperatur und der Innenwandtemperatur bestimmt ist, welche jedoch beide nichtlinear von der Änderungsgeschwindigkeit der Fluidtemperatur (und somit $\dot{u}$) abhängig sind.

Zur Bildung eines allgemeineren Ersatzmodells, das sämtliche Eigenschaften des realen Systems abbilden kann, werden in den folgenden Abschnitten zeitdiskrete und zeitstetige neuronale Netzwerke herangezogen und auf ihre Vorhersagequalität hin untersucht.

6.2.3 Neuronales Netz in diskreter Zeit

Wie in Teil I dieser Arbeit detailliert beschrieben wurde, ist vor dem Training eines diskreten neuronalen Netzwerks zunächst dessen Topologie zu wählen. In diesem Zusammenhang wurde der Lipschitz-Index

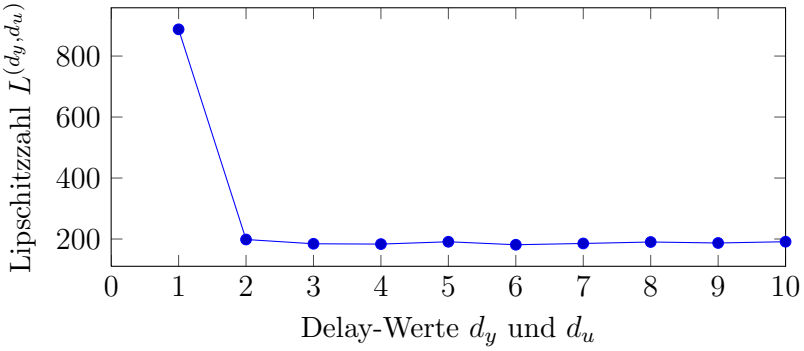


Abbildung 6.5.: Lipschitz-Indizes der Trainingsdaten aus Abbildung 6.2.

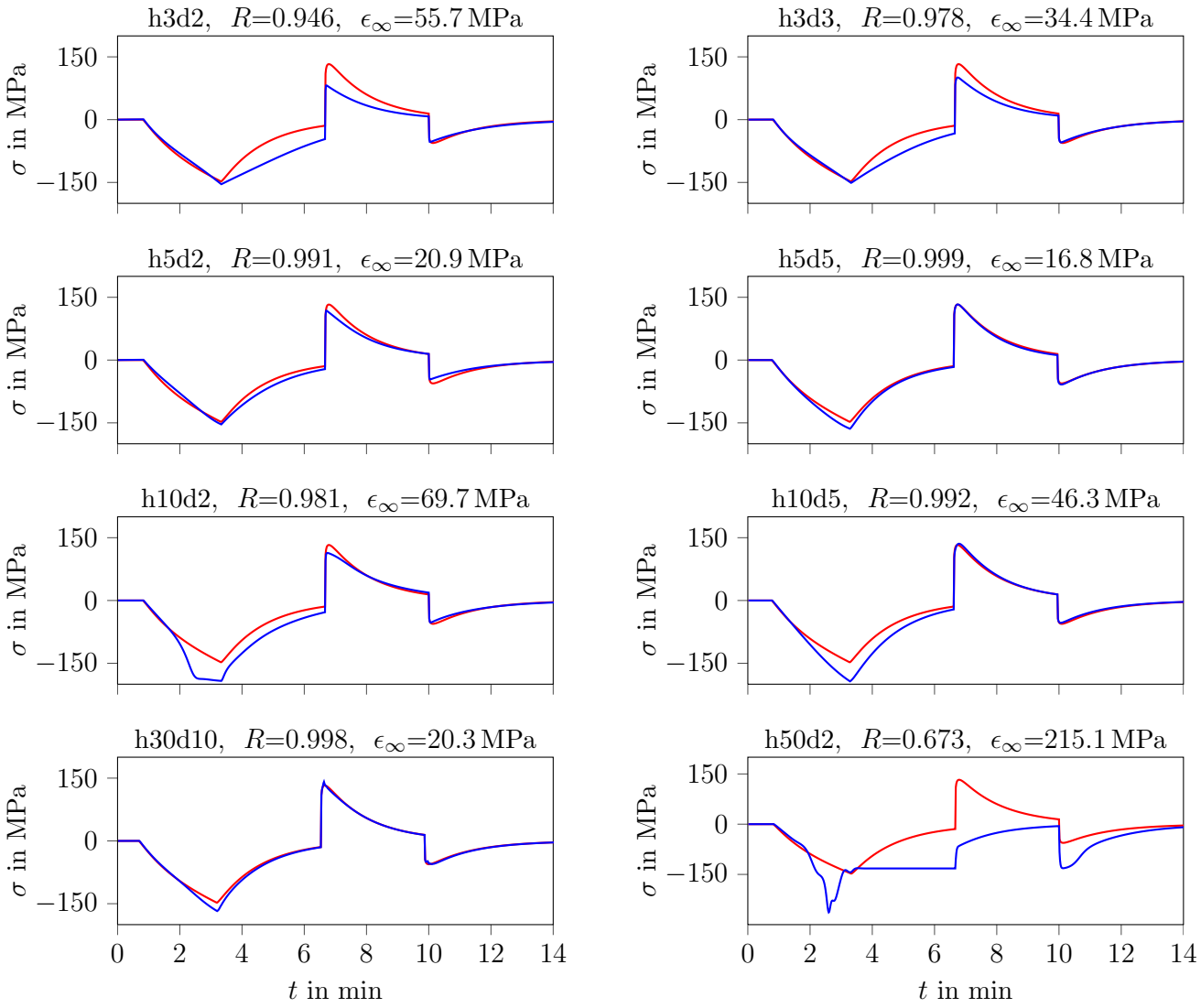


Abbildung 6.6.: Validierungsergebnisse verschiedener NARX-Netze. Verglichen werden die Trainingsdaten (—) mit den Vorhersagen der Netze (—).

eingeführt, der eine Hilfestellung zur Wahl der richtigen Verzögerungswerte d_u und d_y gegeben kann. Bestimmt man diesen für die Trainingsdaten nach Abbildung 6.2, so erhält man die in Abbildung 6.5 dargestellten Ergebnisse. Aus diesen ist einfach ersichtlich, dass gute Verzögerungswerte bei $d_u = d_y = 2$ liegen. Bei diesem Wert ist zu erwarten, dass ein neuronales Netz einerseits ausreichend komplex genug ist, um die wesentliche Systemdynamik hinreichend abzubilden, und andererseits schlank genug gewählt ist, um Overfitting zu vermeiden.

Um dennoch eine möglichst breite Palette unterschiedlicher Netzkomplexitäten zu untersuchen, wurden verschiedene Netzwerke trainiert. Dabei kamen Verzögerungen von 2 bis 10 sowie Anzahlen an Neuronen in der versteckten Schicht von 3 bis 50 zum Einsatz. Dabei wurde aus Gründen der Übersichtlichkeit die Notation hxdy eingeführt, die ein neuronales Netzwerk mit x Neuronen in der versteckten Schicht sowie dem Verzögerungswert y für d_u und d_y bezeichnet. Bezüglich des Trainingsdatensatzes erreichten alle Konfigurationen eine Korrelation über 99 % mit maximalen absoluten Fehlern zwischen 1 und 20 MPa. Die Trainingsergebnisse ausgewählter neuronaler Netzwerke sind dem Anhang A.3 in Abbildung A.1 beigefügt. Wesentlich aussagekräftiger als die Trainingsergebnisse sind jedoch die Vorhersagen der neuronalen Netze bezüglich des Validierungsdatensatzes aus Abbildung 6.3. Diese sind in Abbildung 6.6 dargestellt. Die besten Ergebnisse liegen für die Topologien h5d2 ($n_\eta = 31$), h5d5 ($n_\eta = 61$) und h30d10 ($n_\eta = 661$) vor. Gut erkennbar ist, dass zu geringe Komplexitäten die Systemdynamik nicht richtig auflösen können und zu große Komplexitäten zwar zu einer zusätzlichen Verbesserung bezüglich der Trainingsdaten führen (siehe Abbildung A.1), allerdings die Generalisierung des Netzes darunter leidet, sodass die Vorhersagequalität bezüglich der Validierungsdaten wieder sinkt. Trotz des leicht schlechteren Vorhersageergebnisses wird der Topologie h5d2 der Vorzug gegeben, da diese bei weitaus geringerer Komplexität konkurrenzfähige Ergebnisse liefert. Diese Auswahl ist auch in Hinblick auf die spätere Ableitung eines zeitstetigen Ersatzmodells aus dem neuronalen Netz von Vorteil, da bei der Topologie h5d5 Ableitungen bis hin zur fünften Ordnungen mit entsprechenden Approximationsfehlern in das Problem eingeführt werden müssten.

Um eine Optimierung in stetiger Zeit vornehmen zu können, muss gemäß den Ausführungen in Kapitel 4 zunächst noch ein stetiges Ersatzmodell abgeleitet werden. Hierzu wurde das in Abschnitt 4.4 entwickelte Verfahren verwendet. Bei einer Zeitschrittweite der Trainingsdaten von $\triangle t = 1\,\mathrm{s}$ ergeben sich die Konstanten α, β, δ und κ, die im Anhang in Tabelle A.1 enthalten sind.

6.2.4 Neuronales Netz in stetiger Zeit

Neben einem NARX-Netz in diskreter Zeit sollen zur Bildung eines Ersatzmodells der Mischungsstrecke auch zeitstetige CTRNN-Netze verwendet werden, um ihre Eignung als Ersatzmodelle zu untersuchen. Zum Training des CTRNN-Netzes wurde das Levenberg-Marquardt-Verfahren, wie es in Kapitel 4 formuliert wurde, eingesetzt. Abweichend vom Training der NARX-Netze wurde aus Gründen der Rechenzeitersparnis das Training mit den Validierungsdaten nach Abbildung 6.3 durchgeführt, während die eigentlichen Trainingsdaten nach Abbildung 6.2 zur Validierung herangezogen wurden. Darüber hinaus konnte als Steuerung u nicht die Ventilstellung direkt übernommen werden, da sonst keine implizite Berücksichtigung ihrer Ableitung möglich gewesen wäre. Dieser Nachteil der CTRNN-Netze wurde bereits in Abschnitt 4.3 erläutert. Daher wurde als eigentliche Steuerung die Ableitung der Ventilstellung herangezogen. Diese musste vor dem Training numerisch aus den Trainingsdaten abgeleitet werden.

Abbildung (6.7) zeigt die Trainings- und Validierungsergebnisse dreier CTRNN-Netze. Ersteres entspricht der Form nach den Gleichungen (3.10a) und (3.10b) mit $n_x = 3$. Die beiden anderen Netze entsprechen der Form der Gleichungen (3.7a) und (3.7b) mit $n_x = n_h = 1$ einerseits und $n_x = n_h = 2$ andererseits. Als Ergebnis kann festgehalten werden, dass alle drei CTRNN-Varianten sehr gute Ergebnisse bei geringsten Komplexitäten liefern, was einerseits durch die einfache Systemdynamik und andererseits durch die natürliche, differenzielle Formulierung der Netze erklärbar ist. Auffällig ist der hohe Korrelationswert der Vorhersagen mit den Sollwerten bei gleichzeitig recht hohen maximalen Fehlern ϵ_∞. Dies legt nahe, dass die größeren absoluten Fehler nur auf sehr kurzen Zeitintervallen vorliegen. In

der Tat treten diese nur an den Sprungstellen auf und sind vor allem dadurch erklärbar, dass durch die erwähnte numerische Ableitung der Ventilstellung vor dem Training an den Sprungstellen von u Probleme auftreten, weswegen diese verschmiert werden mussten. Diese zusätzliche Approximation gegenüber dem Originalsystem führte zu einem steilem linearen Verhalten an den Sprungstellen, was besonders gut anhand des Netzes mit $n_x = n_h = 2$ zu beobachten ist. Dadurch kommen die relativ großen Fehler an den Sprungstellen zustande. Aus diesem Grunde sollten, sofern möglich, für zukünftige Anwendungen dieses Verfahrens möglichst keine unstetigen Steuerungen u als Trainingsdaten vorgegeben werden. Die Vorhersagequalität als Ganzes wird von diesem Phänomen jedoch nur wenig beeinflusst. Darüber hinaus ist ohnehin ausgeschlossen, dass dieses Verhalten während der Optimierung zum Tragen kommt, da Sprünge in der Ventilsteuerung aufgrund der Beschränkung (6.1e) ausgeschlossen sind.

Für das weitere Vorgehen wurde das als zweite untersuchte CTRNN-Netz mit $n_x = n_h = 1$ ausgewählt, da es bei den Validierungsdaten die höchste Korrelation und immer noch einen relativ geringen maximalen Absolutfehler aufweist.

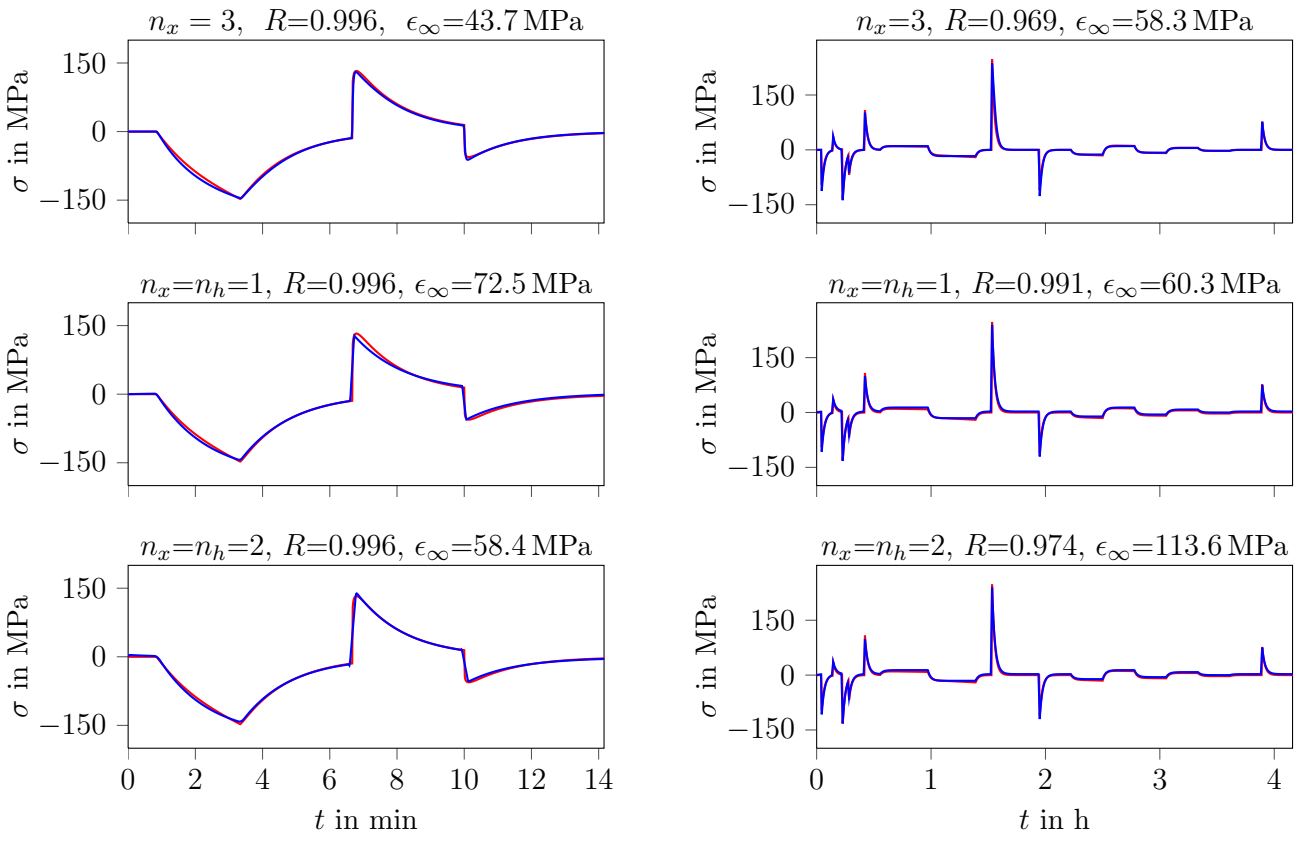


Abbildung 6.7.: Trainings- und Validierungsergebnisse verschiedener zeitstetiger neuronaler Netzwerke. Verglichen werden die Daten des realen Systems (—) mit den Vorhersagen der Ersatzmodelle (—).

6.3 Optimierung der Umschaltsteuerung

Nachdem nun verschiedene Ersatzmodelle vorliegen, sollen diese einer Optimierung unterzogen werden. Hierzu werden einerseits das aus dem NARX-Netz h5d2 stammende und andererseits das CTRNN-Ersatzmodell mit der Komplexität $n_x = n_h = 1$ herangezogen.

Die mathematische Problemformulierung, wie sie in einem OPTIMICA-Skript zu implementieren ist, entspricht den Gleichungen (6.1a)–(6.1k). Zur Lösung des Optimalsteuerungsproblems mit dem Kollokationsverfahren wurde eine sukzessive Verfeinerung der Zeitsegmentierung unter zunehmend schärferen

Toleranzen verwendet. Dabei war es nötig, als eigentliche zu optimierende Steuerung die zweite Ableitung der Ventilposition heranzuziehen, um alle Beschränkungen berücksichtigen zu können.

Abbildung 6.8 illustriert die Ergebnisse der Optimierung. Hier fällt auf, dass auf Basis des CTRNN-Ersatzmodells ein erheblich schnelleres Umschalten der Mischungsstrecke möglich erscheint. Bei näherer Betrachtung der Ergebnisse wird klar, dass das CTRNN-Modell bei gleicher thermischer Spannung gegenüber dem NARX-Modell höhere Änderungsgeschwindigkeiten in der Ventilstellung zulässt, was sich über die recht lange Zeitspanne der aktiven Beschränkung an σ zu einem beachtlichen Zeitvorteil

aufsummiert. So liegt die Endzeit t_f des Optimierungsergebnisses auf Basis des NARX-Modells bei etwa 133 Sekunden, während beim CTRNN-Netz $t_f \approx 97\,\mathrm{s}$ ist. Diese Beobachtung legt den Verdacht nahe, dass (mindestens) eines der beiden Modelle gegenüber dem realen System ungenau sein muss. Dies wird im folgenden Abschnitt 6.4 untersucht, der der Validierung der Ergebnisse gewidmet ist.

6.4 Validierung der Ergebnisse

Die Validierung der Ergebnisse am Originalsystem ist ein wichtiger Schritt, um das Ausmaß der Fehler des Gesamtverfahrens abzuschätzen. Würde das Ergebnis der Optimierung ohne Kontrolle direkt in den realen Prozess eingespeist, so ist nur dann ein richtiges Systemverhalten zu erwarten, wenn in allen Verfahrensschritten, insbesondere aber der Systemidentifikation, verschwindend geringe Fehler gemacht wurden.

Zur Kontrolle der Optimierungsergebnisse wurden diese in das Dymola-Modell, welches ursprünglich zur Erzeugung der Trainings- und Validierungsdaten herangezogen wurde, implementiert. Die entsprechenden Systemantworten zeigt Abbildung 6.9. Hier bestätigt sich der Verdacht aus dem vorigen Abschnitt, dass ein Ersatzmodell offenbar nicht von hinreichender Qualität ist: Die Lösung auf Basis des NARX-Netzes schafft es nicht, die Beschränkung an σ zum schnellen Umschalten der Steuerung voll auszureizen, obwohl dies das Optimierungsergebnis in Abbildung 6.8 suggeriert. Dies ist darauf zurückzuführen, dass das entsprechende Ersatzmodell die Entwicklung der thermischen Spannungen in Abhängigkeit der Steuerung nicht richtig vorhersagt. Im Gegensatz hierzu scheint die Lösung auf Basis des CTRNN-Netzes wirklich optimal zu sein, da sie die vorgegebenen Beschränkungen (6.1e)–(6.1g) zur Gänze ausnutzt. Darüber hinaus fällt bei der optimalen Steuerung auf Basis des NARX-Netzes auf, dass die thermisch induzierten Spannungen gegen Ende der Steuerung (zwischen $t = 100\,\mathrm{s}$ und $t = 120\,\mathrm{s}$) wieder etwas steigen, was auf eine leichte Beschleunigung der Öffnung des Ventils zurückzuführen ist. Außerdem ist ein leichtes anfängliches Überschwingen zu verzeichnen, sodass die Beschränkung an die thermischen Spannungen kurzzeitig um etwa 5 MPa verletzt ist.

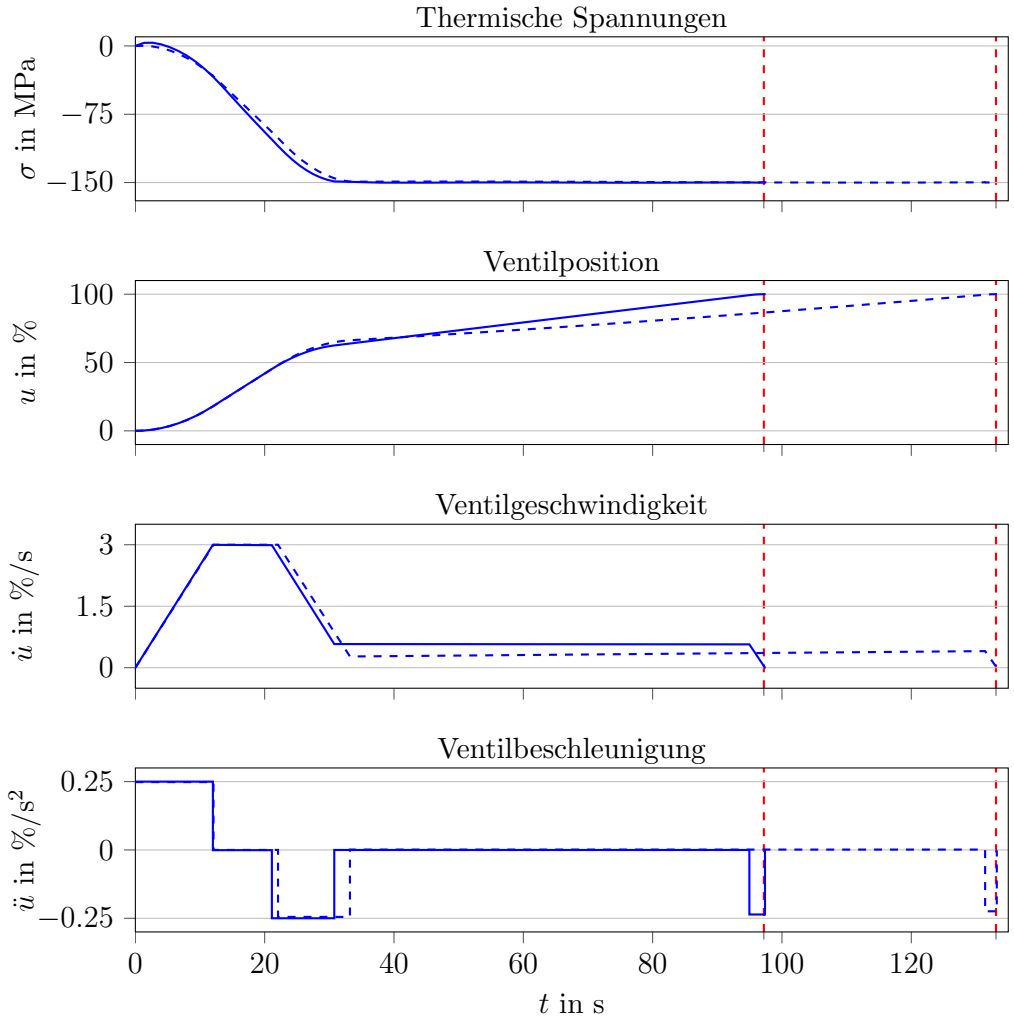


Abbildung 6.8.: Ergebnisse für die optimale Umschaltsteuerung auf Basis eines NARX-Modells (- - -) und eines CTRNN-Modells (——) mit den jeweiligen Endzeiten (- - -).

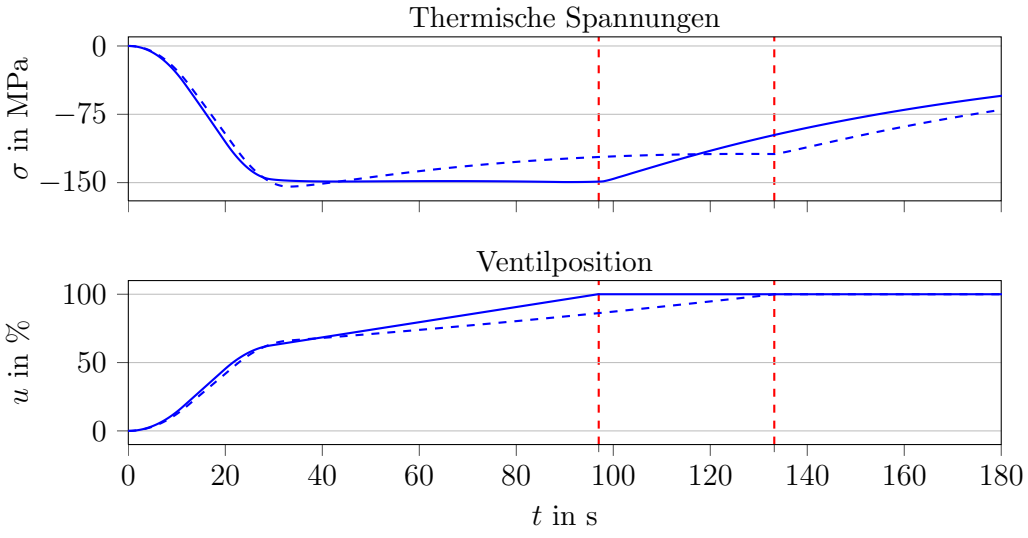


Abbildung 6.9.: Validierungsergebnisse der Umschaltsteuerungen auf Basis des NARX-Netzwerks (- - -) und des CTRNN-Netzes (——) mit den jeweiligen Endzeiten (- - -).

7 Anfahren eines GuD-Prozesses

7.1 Beschreibung der Anlage und des Optimierungsproblems

Gasgefeurte Anlagen, insbesondere kombinierte Gas- und Dampfkraftwerke (GuD), gelten derzeit als die effizientesten und flexibelsten Anlagen auf dem Energiemarkt. Durch ihre hohen Lastwechselgeschwindigkeiten bei gleichzeitig hohen Wirkungsgraden über einen breiten Lastbereich und niedrigen Investitionskosten stellen sie eine gute Möglichkeit dar, um starken Fluktuationen in der Einspeisung regenerativ erzeugter Energie zu begegnen (Bräuer u. Gericke 2014).

Der Warmstart einer derartigen Anlage soll in diesem Abschnitt mittels des im Rahmen dieser Arbeit entwickelten Optimierungsverfahrens hinsichtlich der Anfahrdauer optimiert werden. Hierzu wurde eine GuD-Anlage in Malaysia, welche in der Literatur bereits simulativ untersucht wurde, ausgewählt (Alobaid u. a. 2014a;b). Konkret handelt es sich hierbei um ein Anlagenkonzept mit drei 50 Hz-Gasturbinen des Herstellers General Electric in single shaft-Anordnung und insgesamt drei unterkritischen Abhitzedampferzeugern. Die Dampfturbine, die ebenfalls von General Electric hergestellt wurde, wird von den Abhitzekesseln gespeist, die aufgrund der hohen Rauchgastemperatur der Gasturbine von etwa 628 °C nicht zusätzlich befeuert sind. Jeder Abhitzedampferzeuger arbeitet auf drei Druckniveaus und auf Basis des Zwangumlaufprinzips (Alobaid u. a. 2014a). Die Verschaltung der Heizflächen und der unterschiedlichen Druckniveaus ist aus dem Schaltschema in Abbildung 7.2 ersichtlich.

Zur Optimierung des Anfahrvorgangs wurde ein APROS-Modell des Abhitzedampferzeugers herangezogen, welches auch in den bereits zitierten Untersuchungen zum Einsatz kam (Alobaid u. a. 2014a;b). In diesem Modell wurde ein einzelner Abhitzekessel detailliert abgebildet, wobei die Gasturbine als Randbedingung für den Abgasmassenstrom $\dot{m}_{\mathrm{RG}}$ und die Abgastemperatur T_{RG} zeitabhängig implementiert wurde (Alobaid u. a. 2014a). Um das Modell des Abhitzekessels für die Anwendung des Optimierungsverfahrens aufzubereiten, wurde aus den bekannten Design-Daten und den Simulationsergebnissen eine Gasturbinen-Charakteristik abgeleitet, die in Abhängigkeit ihres Lastzustands u_{GT} die Abgasparameter $\dot{m}_{\mathrm{RG}}$ und T_{RG} bestimmt. Hierbei wurde berücksichtigt, dass gewöhnliche Gasturbinen mittels einer Regelung des Leitschaufelwinkels den Eintrittsquerschnitt in die Gasturbine beeinflussen und somit in letzter Konsequenz die Abgastemperatur über einen breiten Lastbereich von etwa 40 bis 100 % konstant halten können (Dolezal 2001). Zwischen $u_{\mathrm{GT}} = 5\,\%$ und 40 % wurde der Abgasmassenstrom als konstant und die Abgastemperatur als linear abhängig von u_{GT} modelliert (Dolezal 2001). Die aus diesem Ansatz resultierende Gasturbinen-Charakteristik ist in Abbildung 7.1 dargestellt.

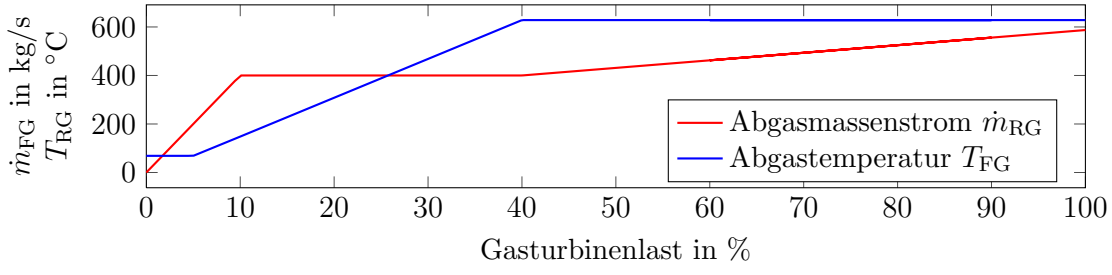


Abbildung 7.1.: Zur Ableitung der Trainingsdaten verwendete Gasturbinen-Charakteristik.

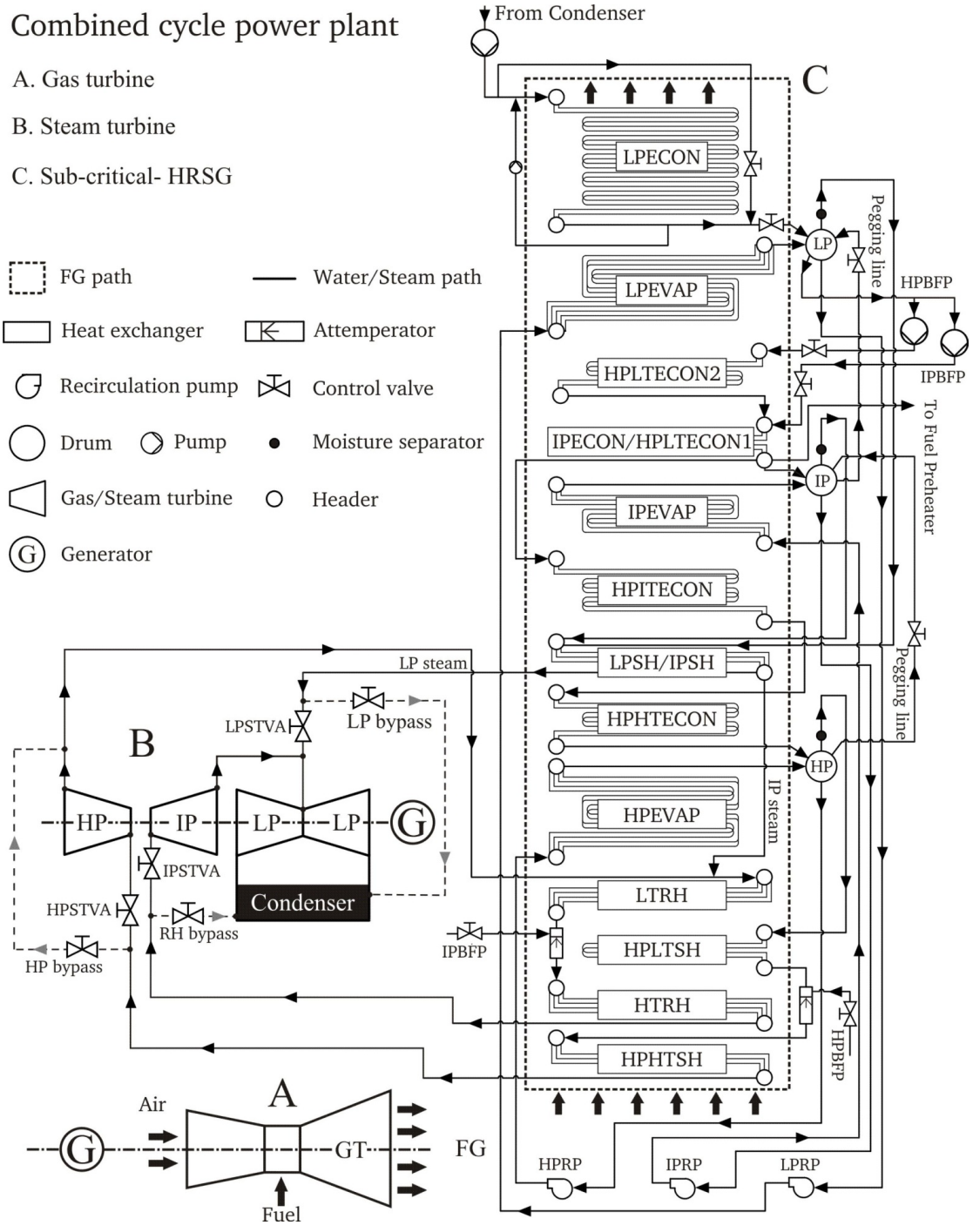


Abbildung 7.2.: Schaltschema des untersuchten GuD-Prozesses (Alobaid u. a. 2014a).

Darüber hinaus wurde das Modell um die Berechnung der Temperaturverteilung in der Wand der Hochdruck-Trommel erweitert, um die in ihr auftretenden thermischen Spannungen in die Untersuchungen mit einbeziehen zu können.

Wie bereits erwähnt, soll der Warmstart der GuD-Anlage hinsichtlich der Anfahrzeit optimiert werden. Dabei sollen als Nebenbedingungen eine Schranke $\sigma_{\max}$ für die maximalen thermischen Spannungen in der Hochdruck-Trommel sowie ein maximaler Lastgradient der Gasturbine von $4\,\%/\mathrm{min}$ zum Einsatz kommen. Letzterer Wert wurde aus der in der Literatur berichteten Anfahrtrajektorie gewonnen (Alobaid u. a. 2014b). Darüber hinaus wird ein monotones Hochfahren der Gasturbine verlangt. Als Anfangswert gilt der Zeitpunkt unmittelbar vor dem Gasturbinen-Start nach mehreren Stunden Stillstand der Anlage. Der Zielzustand ist dadurch gekennzeichnet, dass u_{GT} stationär $100\,\%$ beträgt. Dementsprechend ergibt sich das Optimalsteuerungsproblem zu:

$$\min_{u \in U} \quad J[u(t)] = t_f \tag{7.1a}$$

$$\text{u.B.v.} \quad \dot{x}(t) = f(x(t), u(t), t) \tag{7.1b}$$

$$u_{\mathrm{GT}} \geq 0\,\% \tag{7.1c}$$

$$u_{\mathrm{GT}} \leq 100\,\% \tag{7.1d}$$

$$\dot{u}_{\mathrm{GT}} \leq 4\,\%/\mathrm{min} \tag{7.1e}$$

$$\dot{u}_{\mathrm{GT}} \geq 0\,\%/\mathrm{min} \tag{7.1f}$$

$$|\sigma| \leq \sigma_{\max} \tag{7.1g}$$

Als Anfangs- und Endbedingungen dienen:

$$u_{\mathrm{GT}}(0) = 0\,\% \qquad u_{\mathrm{GT}}(t_f) = 100\,\% \tag{7.1h}$$

$$\dot{u}_{\mathrm{GT}}(0) = 0\,\%/\mathrm{s} \qquad \dot{u}_{\mathrm{GT}}(t_f) = 0 \quad \%/\mathrm{s} \tag{7.1i}$$

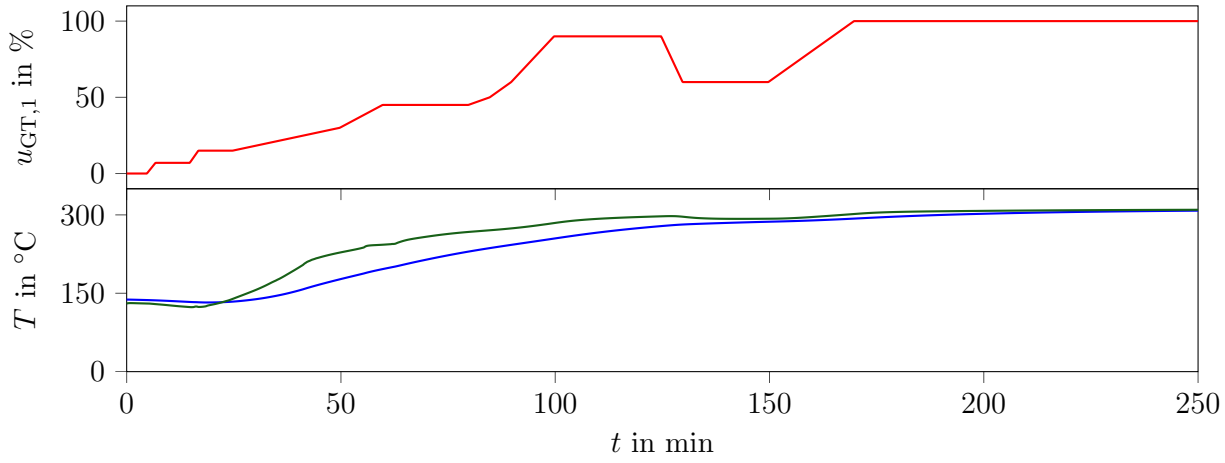
$$\ddot{u}_{\mathrm{GT}}(0) = 0\,\%/\mathrm{s}^2 \qquad \ddot{u}_{\mathrm{GT}}(t_f) = 0 \quad \%/\mathrm{s}^2 \tag{7.1j}$$

Zur Lösung dieser Optimierungsaufgabe soll in den folgenden Abschnitten ein Ersatzmodell erzeugt und die eigentliche Optimierung vorgenommen werden.

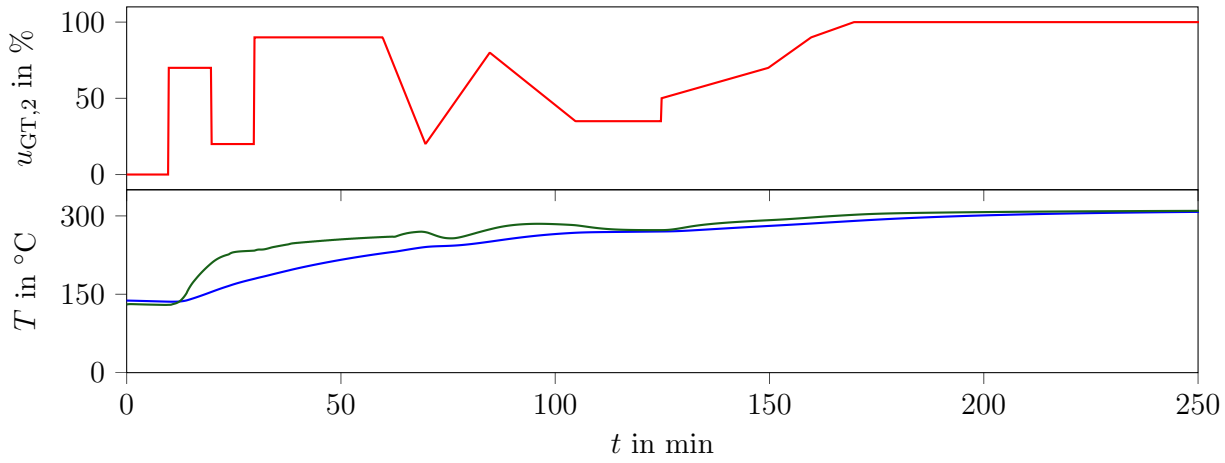
7.2 Ableiten eines Trainingsdatensatzes

Um mit dem Ersatzmodell auch die thermischen Spannungen in der Hochdruck-Trommel berücksichtigen zu können, soll bei diesem Anwendungsbeispiel ein von Kapitel 6 abweichendes Vorgehen verfolgt werden. Anstatt die thermischen Spannungen direkt in die Trainingsdaten mit aufzunehmen, werden nur die zu deren Berechnung nötigen Temperaturen trainiert. Dies sind die mittlere Wandtemperatur T_{avg} und die Wandinnentemperatur T_{i} der Trommel. Dieses Vorgehen hat den Vorteil, dass der Zusammenhang zwischen den Steuerungen und den beiden Temperaturgrößen direkter ist, sodass während des Trainings weniger Abstraktion von den neuronalen Netzen gefordert wird. Nachteilig wirkt sich allerdings aus, dass mit diesem Vorgehen nun zwei hochwertige Ersatzmodelle notwendig sind, um sowohl T_{avg} als auch T_{i} vorhersagen zu können. Aus diesem Ansatz ergibt sich, dass die Trainingsdaten die Größen u_{GT}, T_{avg} und T_{i} umfassen müssen.

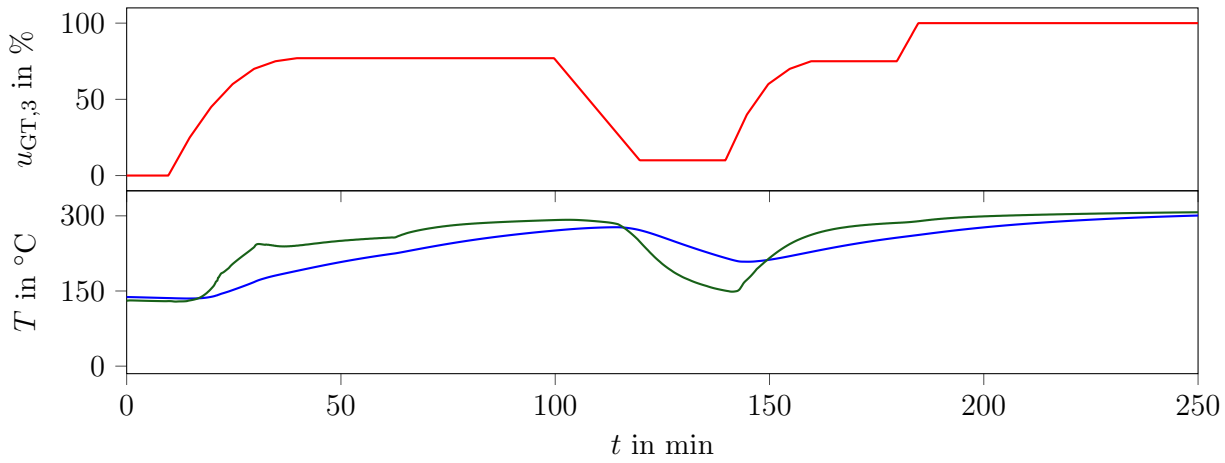
Eine weitere Besonderheit beim Ableiten der Trainingsdaten in diesem Anwendungsfall liegt in der Besonderheit, dass der zu trainierende Anfahrvorgang aus physikalischen Gründen in einer Zeitreihe innerhalb handhabbarer Zeiträume nicht beliebig oft mit den selben Anfangswerten wiederholt werden kann. Aus diesem Grunde wurde der Ansatz verfolgt, drei Trainingsdatensätze zu erstellen, die jeweils einen Anfahrzyklus enthalten. Dabei wurden in jedem der drei Datensätze quantitativ und qualitativ unterschiedliche Verläufe von u_{GT} implementiert. Die damit ermittelten Trainingsdaten sind in Abbildung 7.3 dargestellt. Zum Training wurden diese zu einer Zeitreihe zusammengefügt, wobei der dritte Datensatz die Rolle der Validierungsdaten einnahm.



(a) Erstes Trainingsszenario



(b) Zweites Trainingsszenario



(c) Drittes Trainingsszenario

Abbildung 7.3.: Verschiedene Trainingsszenarien für das Anfahren der GuD-Anlage: Gasturbinen-Last u_{GT} (—), mittlere Trommelwandtemperatur T_{avg} (—) und die Trommelwandinnentemperatur T_{i} (—).

Anhand der Trainingsdaten aus Abbildung 7.3 können nun unterschiedlich komplexe NARX-Netzwerke erzeugt werden. Eine a-priori-Analyse der LIPSCHITZ-Indizes liefert für sowohl für T_i als auch für T_avg einen passenden Verzögerungswert von 2 für d_u und d_y. Trotz dieses Ergebnisses wurden der Vollständigkeit halber auch Netze mit davon abweichenden Verzögerungen erzeugt.

Einige vielversprechende Trainings- und Validierungsergebnisse für die Innenwandtemperatur der Trommel sind in Abbildung 7.4 dargestellt. Wie bereits erwähnt wurden die Trainingsdatensätze 1 und 2 zum reinen Training genutzt, wohingegen Datensatz 3 zur Validierung herangezogen wurde. Es ist erkennbar, dass die etwas milderen Datensätze 1 und 2 insgesamt gut vorhergesagt werden können, während die Vorhersagewerte für den Datensatz 3 Abweichungen zeigt, die für alle dargestellten Netzkomplexitäten sehr ähnlich sind. Diese Abweichungen sind aber vertretbar, da sie insbesondere infolge der schnellen Absenkung von $u_{\mathrm{GT},3}$ nach etwa 100 Minuten auftreten. Für den Optimierungsprozess ist eine derartige Situation nicht zu erwarten, da nach Gleichung (7.1f) kein Absenken von u_GT zulässig ist. Als Ersatzmodell für T_i wurde die Netztopologie h10d2 ausgewählt, da sie bei vergleichsweise geringer Komplexität ein zufriedenstellendes Vorhersageverhalten bietet.

Unabhängig vom Ersatzmodell für T_i ist auch ein entsprechendes Pendant für T_avg zu ermitteln. Im Vergleich zur Innenwandtemperatur der Trommel ist der Trainingsprozess für T_avg als unproblematischer zu bewerten, da die Dynamik der Wandmitteltemperatur wesentlich träger ist. Hinzu kommt, dass auf die Wandmitteltemperatur eventuelle Störungen durch Prozessdetails – beispielsweise das Verhalten im Bypass-Betrieb und der Einspritzregelung – nur sehr gedämpft wirken. Dementsprechend stark sind die Vorhersagen der neuronalen Netze mit den Originaldaten korreliert, wie in Abbildung 7.5 ersichtlich ist. Diese zeigt die Trainings- und Validierungsergebnisse für die Netztopologien h10d2 und h10d5, die beide annähernd gleiche Vorhersagequalitäten bieten. Daher wurde auch hier die Topologie mit nur zweifacher Verzögerung ausgewählt.

Da nun Ersatzmodelle für beide benötigte Temperaturen T_avg und T_i vorhanden sind, lässt sich gemäß

$$\sigma_\mathrm{th} = \frac{\beta \cdot E}{1-\nu} \cdot (T_\mathrm{avg} - T_\mathrm{in}) \tag{7.2}$$

die axiale, thermische Spannung an der Innenwand der Trommel berechnen (Epple u. a. 2012, Al-Zaharnah 2002). Hierin bezeichnen β den linearen Ausdehnungskoeffizienten, E den Elastizitätsmodul und ν die Querkontraktionszahl. An dieser Stelle sollen, wie auch schon beim vorangegangenen Anwendungsfall in Kapitel 6, repräsentativ nur die axialen Spannungen betrachtet werden, da diese in der Nähe des Innenradius ohnehin am größten sind. Darüber hinaus sind geeignete Vergleichsspannungen wie beispielsweise die von-Mises-Spannung in diesem Fall etwa gleich der Axialspannung, da nur geringe Radialspannungen vorliegen und die Tangentialspannungen in ähnlicher Größenordnung wie die Axialspannungen liegen (Al-Zaharnah 2002).

Da die Temperaturen T_avg und T_i als gesonderte Ersatzmodelle vorliegen, kann nun Gleichung (7.2) einfach als zusätzliche Gleichung in das der Optimierung zugrundegelegte Modell eingefügt werden. Aufgrund der Tatsache, dass die Ersatzmodelle für diese Temperaturen anstatt direkt der thermischen Spannung gebildet wurden, lassen sich nun zusätzlich die Stoffwerte β, E und ν als temperaturabhängig modellieren, um die Genauigkeit der Modellierung und damit auch der Optimierungsergebnisse zu erhöhen. Hierzu wurde für das Material der Trommelwand der warmfeste Stahl 10 CrMo 9–10 angenommen, für dessen relevante Stoffwerte folgende Zahlenwertgleichungen bis zu einer Temperatur von 550 °C näherungsweise gelten (VDI 2006):

$$\beta(T) \;=\; \left[11.3 + 10^{-2} \cdot \frac{T}{^\circ\mathrm{C}} - 10^{-5} \cdot \left(\frac{T}{^\circ\mathrm{C}}\right)^2\right] \cdot 10^{-6}/\mathrm{K} \tag{7.3}$$

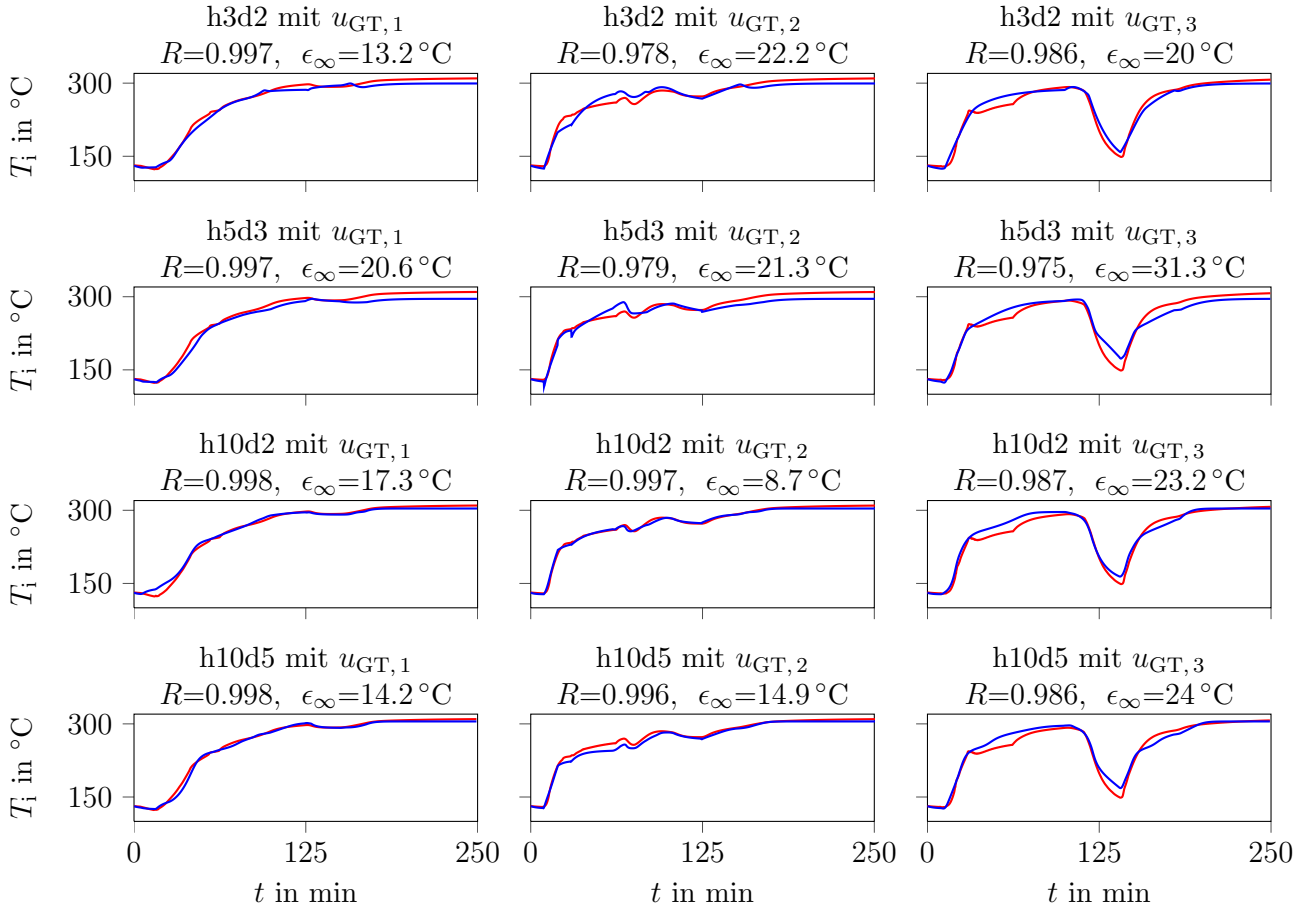


Abbildung 7.4.: Vorhersagen der neuronalen Netzwerke für T_i (—) und dessen wahre Werte (—).

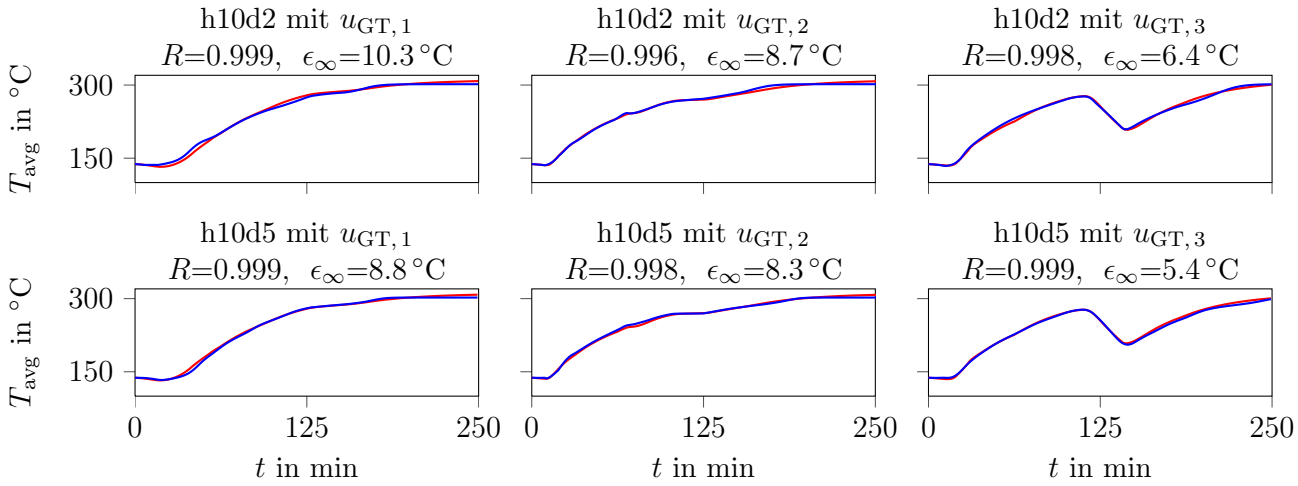


Abbildung 7.5.: Vorhersagen der neuronalen Netzwerke für T_avg (—) und dessen wahre Werte (—).

$$E(T) = \left[216 - 0.07 \cdot \frac{T}{^\circ\mathrm{C}}\right] \cdot \mathrm{kN/mm^2} \tag{7.4}$$

$$\nu = 0.29 \tag{7.5}$$

Das finale Systemmodell, das zur Optimierung verwendet wird, besteht somit aus den beiden Ersatzmodellen für die Trommelinnenwandtemperatur und ihre Mitteltemperatur, die in ein stetiges Ersatzmodell umgewandelt wurden, sowie den Gleichungen (7.2) bis (7.4).

7.4 Optimierung und Validierung

Die Ermittlung der optimalen Steuerung wurde mithilfe des Kollokationsverfahrens in JModelica.org durchgeführt. Hierbei wurden exemplarisch zwei unterschiedliche maximal zulässige Spannungen $\sigma_{\max}$ von einerseits 175 MPa und andererseits 200 MPa festgelegt. Die entsprechenden Ergebnisse für die optimalen Steuerungen u_{GT} und deren reale Systemantworten sind in Abbildung 7.6 dargestellt. Zunächst fällt auf, dass die optimalen Steuerungen durch Kombinationen von linearen Verläufen und Haltephasen gekennzeichnet sind, wobei die linearen Laständerungen stets mit maximal zulässigem Lastgradienten der Gasturbine von $\dot{u}_{\mathrm{GT,\ max}} = 4\,\%/\mathrm{min}$ erfolgen. Darüber hinaus ist ersichtlich, dass beide Steuerungen nur auf einem kurzen Zeitintervall die obere Schranke an die thermische Spannung voll ausreizen können, da aufgrund der Systemdynamik nur kurzzeitig entsprechende Wandtemperaturunterschiede in der Trom-

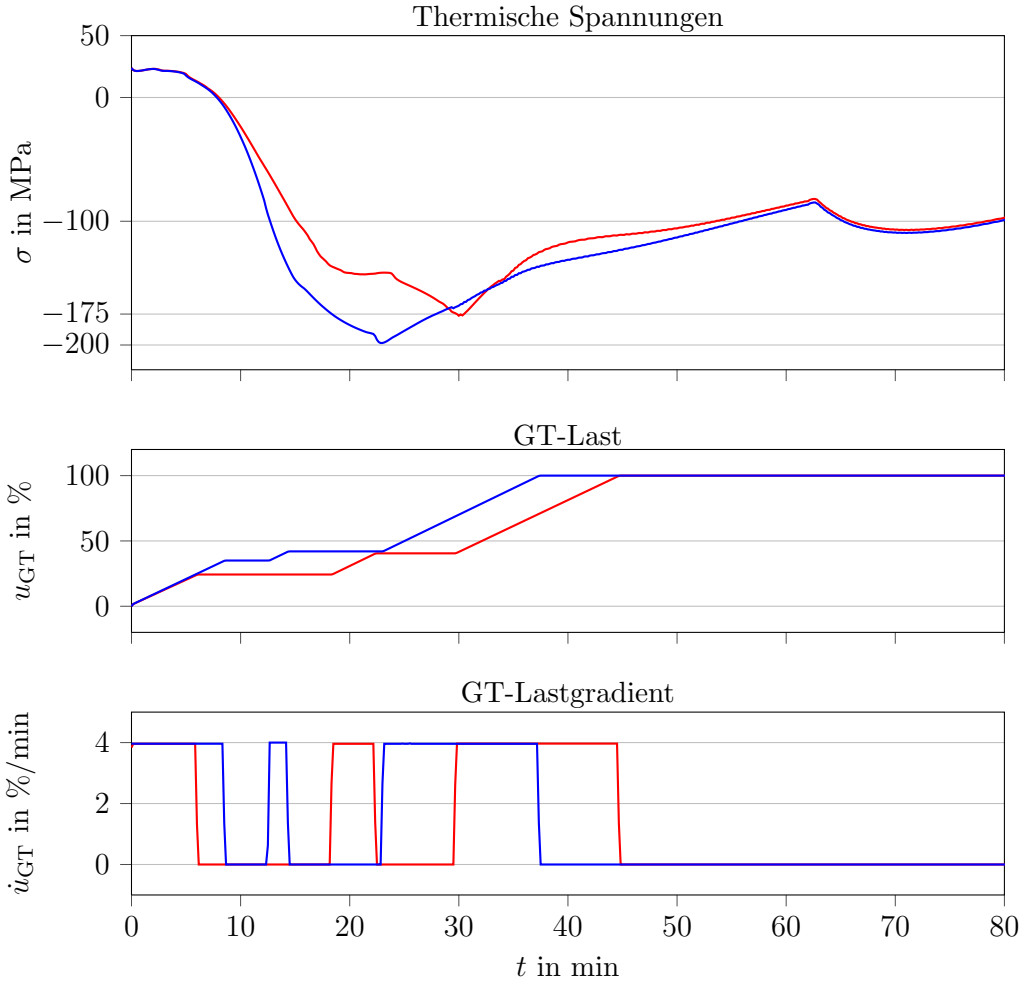


Abbildung 7.6.: Optimierungsergebnisse für die Anfahrsteuerungen mit maximal zulässigen Spannungen von 200 MPa (—) und 175 MPa (—).

mel herrschen. Gegenüber der schnelleren Steuerung mit $\sigma_{\max} = 200\,\mathrm{MPa}$, bei der der Anfahrprozess nach etwa 37 Minuten abgeschlossen ist, weist die langsamere, aber auch schonendere Steuerung einen Zeitnachteil von etwa 8 Minuten auf.

Zuletzt sollen die ermittelten optimalen Steuerungen hinsichtlich ihrer Geschwindigkeit mit einer Referenztrajektorie verglichen werden. Diese wurde in Übereinstimmung mit der postulierten Gasturbinen-Charakteristik nach Abbildung 7.1 aus in der Literatur veröffentlichten Ergebnissen von Anfahrsimulationen ermittelt, wobei an verschiedenen Stellen aufgrund kleinerer Besonderheiten die ermittelte Referenztrajektorie nur approximiert werden konnte (Alobaid u. a. 2014a). Abbildung 7.7 zeigt einen direkten Vergleich der auf diese Weise ermittelten Referenztrajektorie mit den berechneten optimalen Steuerungen. Hierbei wird deutlich, dass die Referenztrajektorie mit einer Endzeit von etwa 145 Minuten einen wesentlich langsameren Anfahrprozess darstellt als die optimierten Steuerungen, obwohl sie beinahe identische Schädigungen der Hochdruck-Trommel bewirkt wie die optimale Steuerung unter der Nebenbedingung $|\sigma| \leq 200\,\mathrm{MPa}$. Von diesen 145 Minuten sind allerdings noch etwa 20 Minuten abzuziehen, die einer anfängliche Spülung des Abhitzedampferzeugers in der Referenztrajektorie entsprechen. Die dennoch hohe zeitliche Diskrepanz zwischen den Referenz- und den optimierten Trajektorien ist überwiegend dadurch zu erklären, dass die Referenztrajektorie nach dem Durchfahren der größten thermischen Spannung eine recht lange Haltephase sowie Lastwechsel mit nicht-maximalen Lastgradienten beinhaltet.

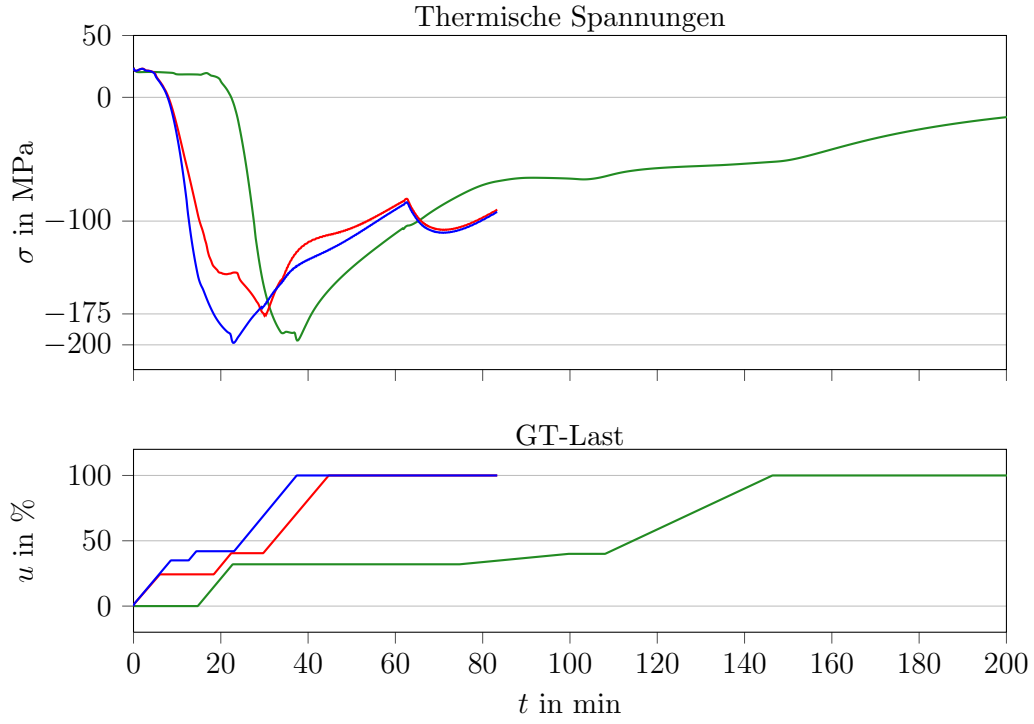


Abbildung 7.7.: Vergleich der Referenztrajektorie (——) mit den optimalen Steuerungen unter den Schranken 200 MPa (——) und 175 MPa (——).

8 Lastwechsel eines Steinkohleblocks

8.1 Anlagen- und Problembeschreibung

Als abschließendes Anwendungsbeispiel für das im Rahmen dieser Arbeit entwickelte Optimierungsverfahren soll ein steinkohlebefeuerter Dampfprozess untersucht werden. Hierzu wurde exemplarisch das Heizkraftwerk Heilbronn der EnBW Kraftwerke AG ausgewählt. Dieses besteht aus sieben Blöcken, von denen einige bereits stillgelegt beziehungsweise in Reserve sind. Insgesamt liefert das Heizkraftwerk eine elektrische Nettoleistung von 1050 MW bei einer maximalen Fernwärmeauskopplung von 390 MW (EnBW Kraftwerke AG 2010). Zur Anwendung des Optimierungsverfahrens soll der Fokus auf den Block 7 gerichtet werden. Dieser wurde bereits 1985 in Betrieb genommen und liefert nach verschiedenen Anlagenoptimierungen im Jahr 2009 eine elektrische Nettoleistung von 800 MW, sodass er leistungsbezogen den weitaus größten Anteil des gesamten Heizkraftwerks darstellt. Die auslegungsgemäßen Frischdampfparameter des Blocks 7 betragen $T_{\mathrm{FD},0} = 545\,^{\circ}\mathrm{C}$ und $p_{\mathrm{FD},0} = 200\,\mathrm{bar}$ (EnBW Kraftwerke AG 2010).

In diesem Kapitel soll eine Optimierung des Lastwechsels von Block 7 von 80 % auf 100 % erfolgen. Hierzu konnte ein bereits bestehendes APROS-Modell verwendet werden, welches am Institut für Energiesysteme und Energietechnik (EST) im Rahmen eines Kooperationsprojektes mit Alstom Power Service entwickelt wurde und die Anlage und ihre Regelungen auf einer hohen Detailebene abbildet. Eine Analyse des Modells in Hinblick auf die Anwendung des Optimierungsverfahrens ergab, dass die Anlagenregelungen einen Laständerungsbefehl im Wesentlichen auf zwei Ebenen umsetzen. Einerseits wird zur Änderung der Brennstoffzufuhr aus dem Lastbefehl ein Stellsignal u_{Zuteil} für die Drehzahl des Zuteilers für jede der vier Steinkohlemühlen abgeleitet. Dieses ist im Viermühlenbetrieb, der im hier interessierenden Leistungsbereich gegeben ist, für alle vier Zuteiler identisch. Andererseits wird in Abhängigkeit von der Last der Speisewassermassenstrom geregelt, um der geänderten Brennstoffzufuhr gerecht zu werden. Hierzu wird ein Regelventil mit dem Signal u_{SPAT} gestellt, das den Dampfzustrom in eine Speisepumpenantriebsturbine (SPAT) bestimmt, wodurch direkt die Speisepumpenleistung und somit der Speisewassermassenstrom beeinflusst wird. Neben diesen beiden, für den Lastwechsel zentralen Regel- und Rechenschaltungen existiert noch eine Vielzahl zusätzlicher Regelungen. Hier ist beispielsweise die Dampftemperaturregelung mittels Einspritzkühlern zu nennen, die versucht, die Frischdampftemperatur T_{FD} im oberen Lastbereich auf dem Auslegungsniveau von $T_{\mathrm{FD},0} = 545\,^{\circ}\mathrm{C}$ zu halten.

Um die Anlage nun einer Steuerung von außen zugänglich zu machen, war es nötig, die entsprechenden Regelungen für das SPAT-Ventil und die Zuteilerdrehzahl im Modell zu überbrücken. Sämtliche andere Regelkreise wurden jedoch im Originalzustand belassen.

Die Optimierung des Lastwechsels soll derart erfolgen, dass er schnellstmöglich vonstatten geht, wobei minimale und maximale Absolutwerte von u_{SPAT} und u_{Zuteil} sowie maximale Änderungsraten dieser Größen einzuhalten sind. Deren entsprechende Grenzwerte wurden aus den im APROS-Modell regelungstechnisch vorgegebenen Minimal- und Maximalwerten sowie deren maximalen Gradienten ermittelt. Als weitere Beschränkung an den Optimierungsprozess soll die Frischdampftemperatur dienen, deren Abweichung von der stationären Temperatur von $T_{\mathrm{FD},0} = 545\,^{\circ}\mathrm{C}$ maximal $\triangle T_{\mathrm{FD,zul}}$ betragen darf. Für diese maximal zulässige Abweichung wurden die Werte 3, 5 und 15 K herangezogen. Aufgrund der Tatsache, dass im oberen Lastbereich infolge der Dampftemperaturregelung nur relativ kleine Änderungen in der Dampftemperatur auftreten, ist eine Beschränkung an die thermischen Spannungen in dickwandigen Bauteilen nur von relativ geringer Bedeutung. So ergab sich in Voruntersuchungen, dass auftretende thermische Spannungen beispielsweise im dickwandigen Zyklon nach dem Verdampfer des Dampferzeugers maximale Beträge von etwa 50 MPa erreichen. In den Sammlern der Überhitzer ist dieser Wert

noch wesentlich geringer, da dort bereits die Einspritzkühler einer zu starken Änderung der Dampftemperatur entgegenwirken. Zu Beginn der Steuerung sei ein stationärer Zustand bei 80 % Last gegeben, der durch eine erzeugte Bruttoleistung von 664 MW, Drehzahlsignalen von 66 % sowie der Position des SPAT-Ventils von 21 % gegeben ist. Diese Werte wurden direkt mit dem zur Verfügung stehenden APROS-Modell ermittelt. Diesen Ausführungen entsprechend ergibt sich eine mathematische Formulierung des Optimierungsproblems wie folgt[1]:

$$\min_{u \in U} \quad J[u(t)] = \int_0^{t_f} \left(x_1(t) - 800\,\mathrm{MW}\right)^2 \,\mathrm{d}t \tag{8.1a}$$

$$\text{u.B.v.} \quad \dot{x}(t) = f(x(t), u(t), t) \tag{8.1b}$$

$$u_{\mathrm{SPAT}} \geq 0\,\% \tag{8.1c}$$

$$u_{\mathrm{SPAT}} \leq 100\,\% \tag{8.1d}$$

$$|\dot{u}_{\mathrm{SPAT}}| \leq 10\,\%/\mathrm{s} \tag{8.1e}$$

$$u_{\mathrm{Zuteil}} \geq 50\,\% \tag{8.1f}$$

$$u_{\mathrm{Zuteil}} \leq 90\,\% \tag{8.1g}$$

$$|\dot{u}_{\mathrm{Zuteil}}| \leq 2.5\,\%/\mathrm{min} \tag{8.1h}$$

$$T_{\mathrm{FD}} \leq 545\,^{\circ}\mathrm{C} + \triangle T_{\mathrm{FD,\,zul}} \tag{8.1i}$$

$$T_{\mathrm{FD}} \geq 545\,^{\circ}\mathrm{C} - \triangle T_{\mathrm{FD,\,zul}} \tag{8.1j}$$

mit den Anfangs- und Endbedingungen

$$T_{\mathrm{FD}}(0) = 545\ ^{\circ}\mathrm{C} \qquad T_{\mathrm{FD}}(t_f) = 545\ ^{\circ}\mathrm{C} \tag{8.1k}$$

$$P_{\mathrm{el}}(0) = 664\ \mathrm{MW} \qquad P_{\mathrm{el}}(t_f) = 800\ \mathrm{MW} \tag{8.1l}$$

$$u_{\mathrm{SPAT}}(0) = 21\ \ \% \tag{8.1m}$$

$$u_{\mathrm{Zuteil}}(0) = 66\ \ \% \tag{8.1n}$$

Um das Optimierungsproblem (8.1a)–(8.1n) zu lösen, ist zunächst das nicht explizit bekannte Systemmodell $f(x, u, t)$ durch ein geeignetes Ersatzmodell $\hat{f}$ zu approximieren. Hierauf wird in den folgenden Abschnitten eingegangen.

8.2 Trainingsdaten zur Erzeugung eines Ersatzmodells

Um ein neuronales Netzwerk zu einem geeigneten Ersatzmodell zu trainieren, sind nach den Ausführungen in Teil I dieser Arbeit Trainingsdaten erforderlich. Im Gegensatz zu den zwei zuvor untersuchten Anwendungsbeispielen ist das hier vorliegende Problem des Lastwechsels nicht mehr durch bloß eine Eingangs- und eine Ausgangsgröße bestimmt. Vielmehr sind einerseits die zwei Eingangsgrößen u_{SPAT} und u_{Zuteil} und andererseits die zwei Ausgangsgrößen P_{el} und T_{FD} Teil des Problems, sodass insgesamt eine höhere Dimensionalität der Trainingsdaten vorliegt. Dadurch, dass die Lastwechseldynamik von kohlebefeuerten Kraftwerken meist relativ träge ist, musste zudem ein zeitlich relativ langer Trainingsdatensatz erzeugt werden, um die Wirkungen der beiden Steuerungen einerseits unabhängig voneinander und andererseits in einem gewissen Verhältnis zueinander in die Trainingsdaten einfließen zu lassen. Die entsprechenden Trainingsdaten zeigt Abbildung 8.1.

Zur Validierung der trainierten neuronalen Netzwerke werden zweierlei Datenquellen herangezogen. Einerseits wird der Trainingsdatensatz blockweise in Trainings- und Validierungsdaten aufgespalten, wobei die Trainingsdaten einen Anteil von 70 % an den Gesamtdaten haben. Andererseits wurden aus dem ursprünglichen APROS-Modell, in dem die Regelungen noch nicht überbrückt wurden, der zeitliche Verlauf der Problemgrößen für einen geregelten Referenzlastwechsel von 80 auf 100 % Last gewonnen. Diese dienen als zusätzlicher Validierungsdatensatz, sodass insgesamt eine breite Datenbasis vorhanden ist.

[1] Zur Wahrung der mathematischen Korrektheit der Notation seien $x_1 := P_{\mathrm{el}}$ und $x_2 := T_{\mathrm{FD}}$ definiert.

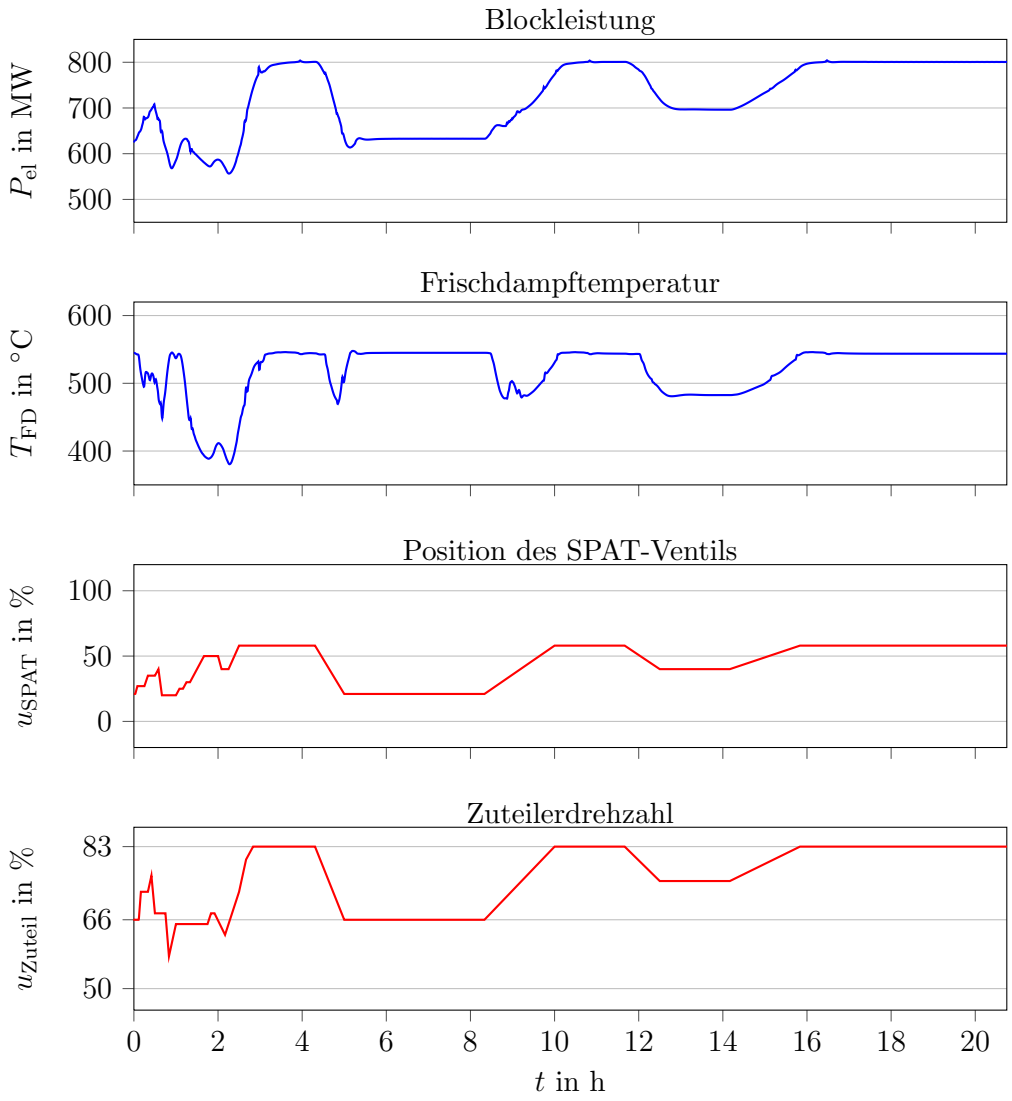


Abbildung 8.1.: Trainingsdaten zur Ermittlung eines Ersatzmodells für die Anlage Heilbronn Block 7. Die Blockleistung und die Frischdampftemperatur bilden die Ausgangsgrößen (——), die Position des SPAT-Ventils sowie die Zuteilerdrehzahl die Eingangsgrößen (——).

8.3 Ermittlung eines Ersatzmodells

Mit den Trainingsdaten nach Abbildung 8.1 wurden verschiedene zeitdiskrete NARX-Netze in unterschiedlichen Komplexitätsvarianten trainiert und validiert. Bei allen trainierten Netzen wurden Verzögerungen von $d_u = d_y = 2$ gewählt, welche mittels einer Analyse der Lipschitz-Indizes als vielversprechende Verzögerungswerte identifiziert wurden. Die Vorhersagequalität der verschiedenen neuronalen Netze in Bezug auf die Trainings- und Validierungsdaten sind in Abbildung 8.2 dargestellt.

Konkret wurden Netze mit 5, 10, 15 und 20 Neuronen in der versteckten Schicht trainiert. Zunächst ist festzustellen, dass für alle untersuchten Komplexitäten die Qualität der Netze in Bezug auf die Trainigsdaten als sehr gut und nahezu unabhängig von den untersuchten Netzkomplexitäten zu bewerten ist. Bezieht man die Validierungsergebnisse in die Bewertung mit ein, so fällt auf, dass das Netz h10d2 die Vorhersage mit der besten Korrelation R und der geringsten maximalen Abweichung ϵ_∞ liefert, die mit 3 MW im Bereich von etwa 0.4 % liegt. Die Netze mit weniger und mehr versteckten Neuronen liefern schlechtere Ergebnisse, da sie entweder nicht komplex genug sind, um die Systemdynamik richtig abzubilden, oder durch eine zu hohe Komplexität einen Verlust an Generalisierung erleiden.

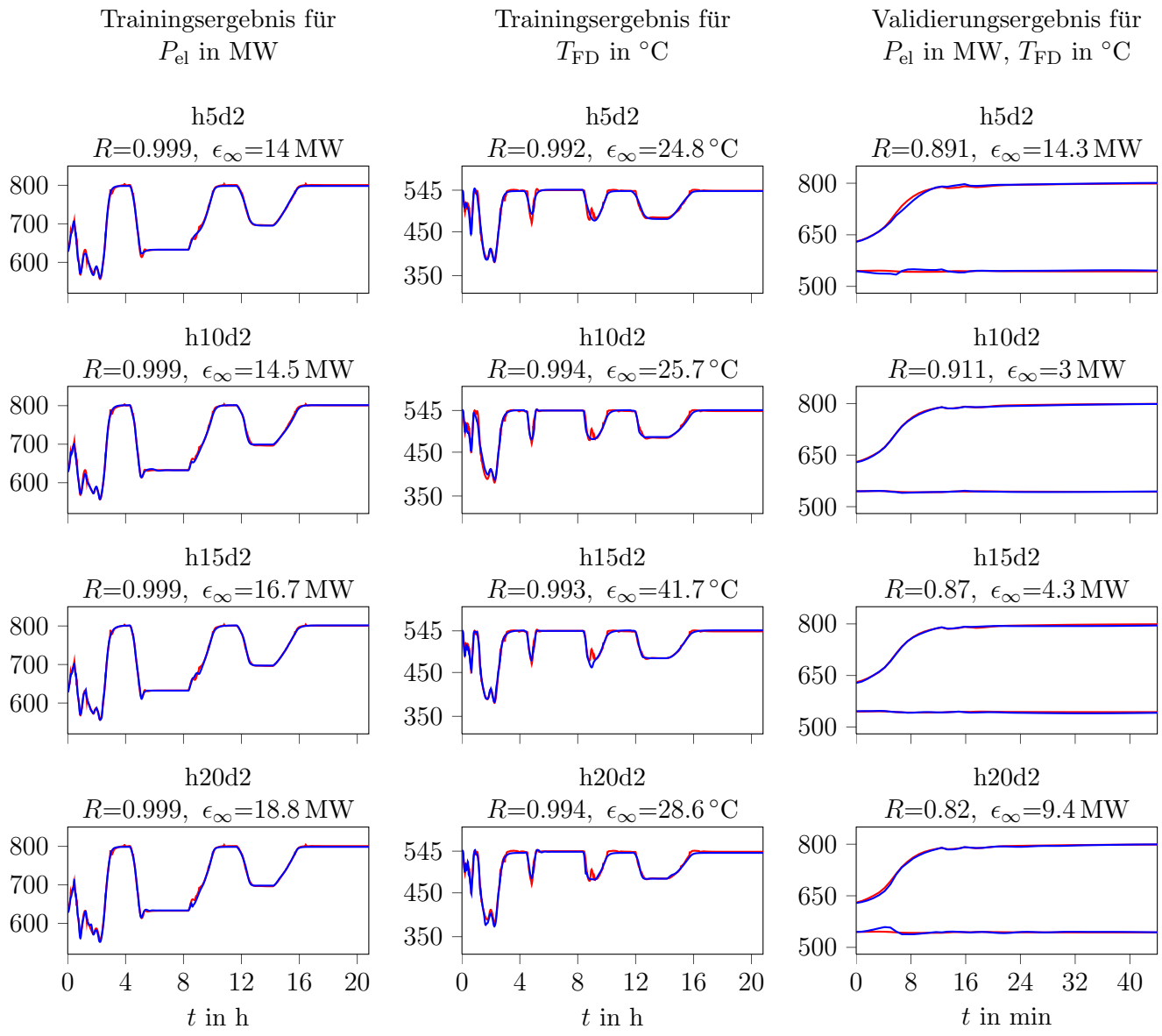


Abbildung 8.2.: Trainings- und Validierungsergebnisse verschiedener NARX-Netzwerke. Verglichen werden die jeweiligen Vorhersagen der Netze (—) und die wahren Werte (—).

In den Vorhersagen aller Netze für T_{FD} fällt auf, dass kurzfristige, starke Abweichungen in der Frischdampftemperatur von den Netzen in geglätteter Weise approximiert werden. Dies ist eine direkte Folge der Generalisierung dieser Netzwerke. Anstatt den genauen Verlauf der Frischdampftemperatur, der in hohem Maße vom Verhalten der nicht abgebildeten Einspritzregelung abhängt, nachzubilden, vermögen die Netze trotz ihrer vergleichsweise geringen Komplexitäten die entsprechenden Werte von T_{FD} in Absolutwert und Verlauf zufriedenstellen abzubilden. Ein ähnlicher Effekt tritt auch in der Vorhersage der elektrischen Leistung auf, wobei diese generell eine wesentlich höhere Trägheit gegenüber den Steuerungen und somit störungsärmeres Verhalten aufweist.

8.4 Berechnung, Diskussion und Validierung der Optimierungsergebnisse

Mit dem im vorstehenden Abschnitt identifizierten Systemmodell wurde eine eine Lösung des Optimierungsproblems (8.1a)–(8.1n) berechnet. Hierzu kam im Gegensatz zu den vorigen beiden Anwendungsbeispielen der Kapitel 6 und 7 statt der zeitstetigen Optimierung mittels des Kollokationsverfahrens eine

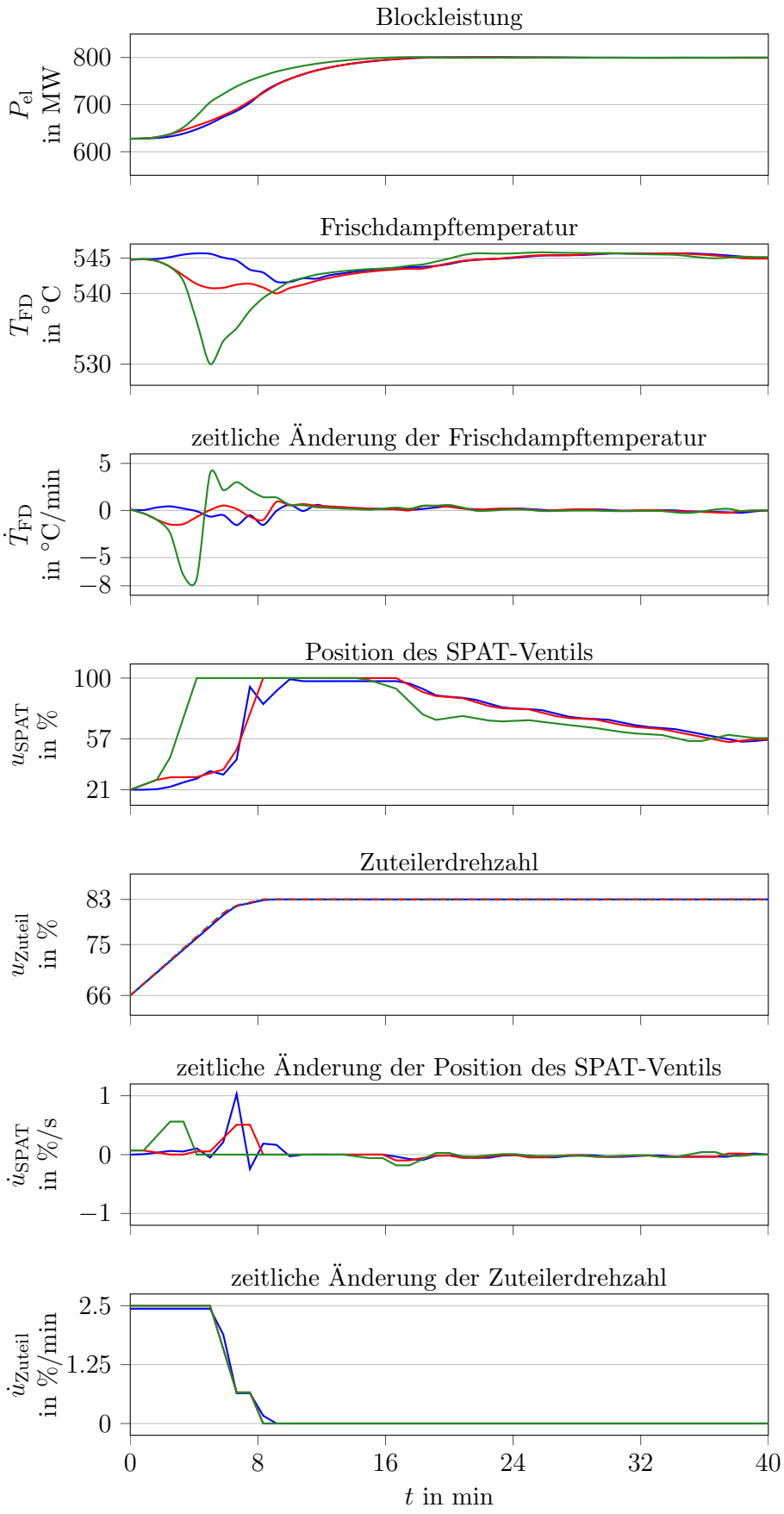


Abbildung 8.3.: Ergebnisse für die optimale Steuerung des Zuteilers und des SPAT-Ventils für zulässige Änderungen $\triangle T_{\mathrm{FD,zul}}$ der Frischdampftemperatur von 3 K (—), 5 K (—) und 15 K (—).

zeitdiskrete Optimierung mit dem direkten Schießverfahren zum Einsatz. Wie in Kapitel 4 detailliert erläutert wurde, kann diese Optimierungsaufgabe in MATLAB mithilfe von bestehenden Algorithmen gelöst werden. Hierzu wurden die beiden Steuerungen in jeweils 20 Zeitsegmente unterteilt. Dementsprechend waren insgesamt 40 Parameter unter den entsprechenden Nebenbedingungen zu optimieren, was mit dem SQP-Verfahren `fmincon` erreicht wurde.

Die auf diese Weise ermittelten Ergebnisse für die optimalen Steuerungen u_{SPAT} und u_{Zuteil} des SPAT-Ventils sowie des Zuteilers sind in Abbildung 8.3 illustriert. Hierin werden die Ergebnisse für die drei untersuchten zulässigen Abweichungen der Frischdampftemperatur von 3, 5 und 15 K farblich unterschieden. So fällt für die erzeugte elektrische Bruttoleistung zunächst auf, dass sie für die beiden stärker restringierten Fälle mit nahezu gleicher Geschwindigkeit auf den Zielwert erhöht werden kann, während für $\triangle T_{\mathrm{FD,\,zul}} = 15\,\mathrm{K}$ eine schnellere Leistungserhöhung erzielt werden kann. Diese erreicht den Zielwert zwar etwa zur selben Zeit wie die stärker beschränkten Varianten, erzielt aber während des Lasterhöhungsprozesses eine signifikant höhere Leistung.

Betrachtet man weiter die optimale Steuerung für den Zuteiler, so fällt auf, dass dieser mit einem maximal zulässigen Gradienten von 2.5 %/min auf einen stationären Endwert von 83 % angehoben wird. Demzufolge wird die Brennstoffzufuhr schnellstmöglich auf das der Ziellast entsprechende Niveau erhöht. Um die Balance zwischen Brennstoffzufuhr und Speisung des Dampferzeugers aufrechtzuerhalten und in letzter Konsequenz die Frischdampftemperatur innerhalb des zulässigen Bands $\left[545\,^{\circ}\mathrm{C} - \triangle T_{\mathrm{FD,\,zul}}\,,\ 545\,^{\circ}\mathrm{C} + \triangle T_{\mathrm{FD,\,zul}}\right]$ zu halten, muss der Speisewassermassenstrom über ein Öffnen des SPAT-Ventils erhöht werden. Dabei ergibt sich für die optimale Steuerung des SPAT-Ventils eine Übersteuerung, wodurch die Speisung des Dampferzeugers schnell erhöht wird, was ein Absenken der Frischdampftemperatur bewirkt, wobei jedoch die Einhaltung der Temperaturbeschränkung gewährleistet ist. Nach einer gewissen Zeitspanne dieser vollen Öffnung des SPAT-Ventils wird u_{SPAT} gleichmäßig auf einen stationären Endwert von $u_{\mathrm{SPAT}}(t_f) = 57\,\%$ gesenkt. Darüber hinaus stellt die Einschränkung der Öffnungsgeschwindigkeit des SPAT-Ventils offenbar keine Beeinträchtigung der Lastwechselgeschwindigkeit dar, da die optimalen Steuerungen diese Beschränkung nur zu etwa 10 % ausreizen. Dies ist auch der Grund für die schnellstmögliche Erhöhung der Zuteilerdrehzahl. Da diese in ihrer Dynamik wesentlich stärker beschränkt ist, muss sie zur Erlangung von Zeitoptimalität mit maximalem Gradienten erhöht werden, wohingegen das wesentlich schnellere SPAT-Ventil die dadurch entstehenden Störungen kompensiert.

Wie aus Abbildung 8.3 hervorgeht, werden die jeweiligen Beschränkungen an die Frischdampftemperatur in allen Fällen eingehalten. Die größte Abweichung von T_{FD} erzeugt hierbei das vollständige Öffnen des SPAT-Ventils, wodurch eine mehr oder weniger starke Absenkung der Frischdampftemperatur entsteht. Der Betrag dieser Absenkung ist jedoch bestimmend für die Lastwechselgeschwindigkeit, da ein schnelles, übersteuertes Öffnen des SPAT-Ventils auch eine schnelle Erhöhung des Speisewasserstroms auf den Volllast-Zustand ermöglicht.

Betrachtet man die zeitliche Änderung der Frischdampftemperatur, $\dot{T}_{\mathrm{FD}}$, so lässt sich beobachten, dass die höchsten Abkühlungs- und Aufheizungsraten des Frischdampfes in dem Zeitraum vorkommen, in dem u_{SPAT} übersteuert. So liegt beispielsweise für den schnellen Lastwechsel, der eine Abweichung der Frischdampftemperatur von 15 K zulässt, kurzzeitig eine Abkühlungsrate von 8 K/min vor, die relevant für die Ausbildung thermischer Spannungen wie etwa im Turbinenrotor ist.

Abschließend soll im Rahmen einer Validierung der Optimierungsergebnisse auch ein Vergleich mit einer Referenztrajektorie gezogen werden. Hierzu wurde der zeitliche Verlauf der Problemgrößen im geregelten (und nicht gesteuerten) Referenzfall herangezogen. Diese Daten dienten bereits als Validierungsdatensatz für die neuronalen Netze. Eine Analyse der Referenztrajektorie ergab maximale Abweichungen der Frischdampftemperatur von $\triangle T_{\mathrm{FD,\,zul}} \approx 3\,\mathrm{K}$, sodass die Laständerungsgeschwindigkeit des Referenzfalls gut vergleichbar mit dem Optimierungsergebnis für die entsprechende zulässige Temperaturabweichung ist. Die Ergebnisse dieses Vergleichs sind in Abbildung 8.4 dargestellt. Hier fällt auf, dass die Geschwindigkeiten der Laständerung nahezu identisch sind und sowohl im Referenzfall als auch unter

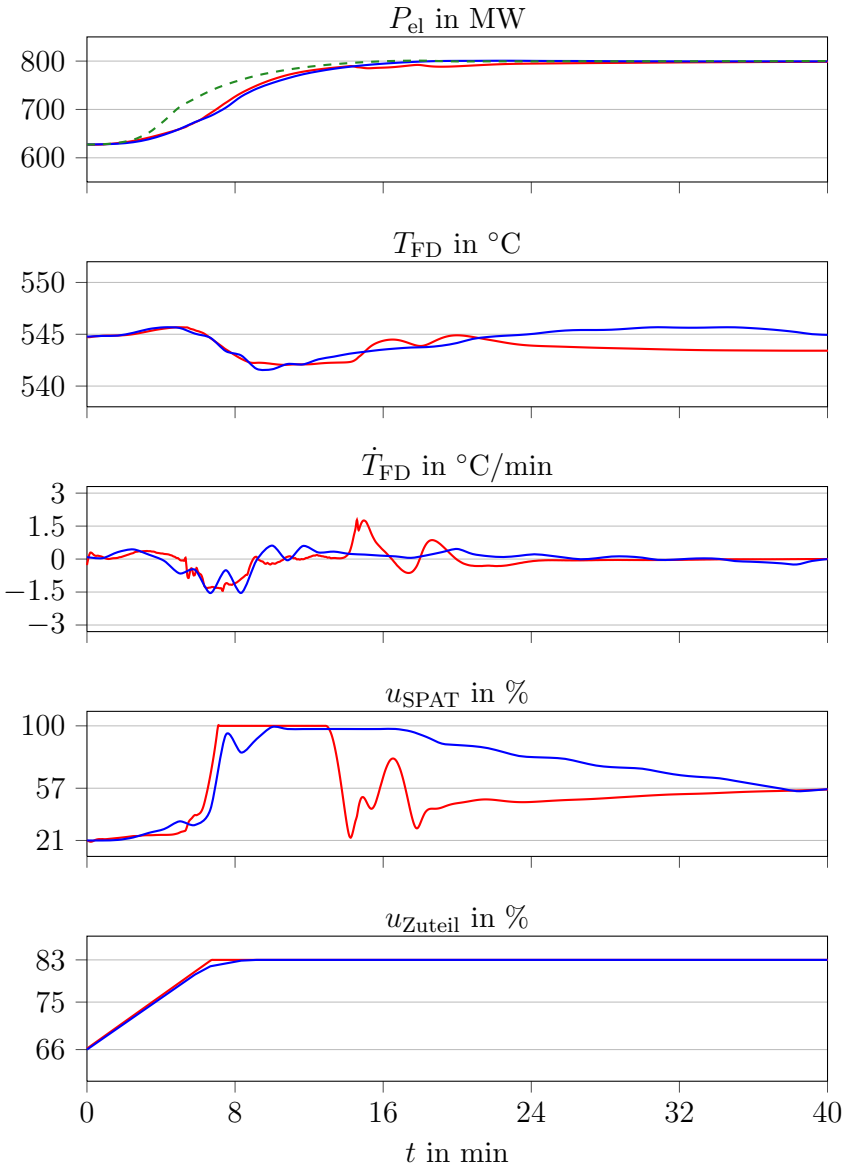


Abbildung 8.4.: Vergleich des Optimierungsergebnisses (——) mit der Referenztrajektorie (——). Zusätzlich ist der mögliche Zeitgewinn für $\triangle T_{\mathrm{FD,\,zul}} = 15\,\mathrm{K}$ angedeutet (- - -).

der optimierten Steuerung die Frischdampftemperatur innerhalb des zulässigen Bands bleibt. Auch die thermische Belastung der dickwandigen Bauteile, die anhand der zeitlichen Änderungsrate der Frischdampftemperatur bewertet sei, ist in beiden Fällen sehr ähnlich. Vor diesem Hintergrund ist als Ergebnis festzuhalten, dass in Bezug auf die Laständerungsgeschwindigkeit die implementierte Leistungsregelung des Blocks 7 des Heilbronner Heizkraftwerks sehr nahe an einem Zeitoptimum ist.

Lässt man anstatt von $\triangle T_{\mathrm{FD,\,zul}} \approx 3\,\mathrm{K}$ eine Abweichung von 15 K zu, so lässt sich der in Abbildung 8.4 angedeutete Zeitgewinn erzielen. Wie bereits erwähnt, erreicht dieser zwar den stationären Endwert der elektrischen Bruttoleistung von 800 MW zwar etwa zum gleichen Zeitpunkt, jedoch geht der Lastwechselvorgang in der Anfangsphase mit stärkeren Gradienten vonstatten, sodass während des gesamten Lastwechsels höhere Leistungen in das Netz eingespeist werden als im Referenzfall.

9 Fazit

Stromerzeugungsanlagen sehen sich ständig steigenden Anforderungen an die Wirtschaftlichkeit und Flexibilität gegenübergestellt. Aus diesem Grund ist die optimale Gestaltung und Führung der Prozesse wünschenswert. Hierbei stellt sich die jedoch Frage, wie derartige optimale Prozessführungen im Sinne einer optimalen Steuerung berechnet werden können.

Es gibt zwar eine gut erforschte Theorie zur Berechnung optimaler Steuerungen, die jedoch nur anwendbar ist, wenn ein explizites Systemmodell vorliegt. Dies ist aber in der Realität oftmals nicht gegeben, weder bei physisch vorhandenen Anlagen, noch bei Simulationsmodellen. Letztere implementieren implizit zwar Modellgleichungen, die aber in der Regel wegen des Black-Box-Verhaltens der Simulationsumgebungen nicht vom Benutzer extrahierbar und weiterverwendbar sind. Gegenstand dieser Arbeit war es dementsprechend, ein Gesamtverfahren zu entwickeln, das diese Lücke schließt und eine Anwendung der Theorie der optimalen Steuerungen auf unbekannte Systeme erlaubt.

Nachdem in einer grundlegenden Konzeptphase unterschiedliche Vorgehensweisen verglichen und gegeneinander abgewogen wurden, stand im Mittelpunkt der weiteren Bemühungen ein Ansatz, bei dem das unbekannte System zunächst durch ein mathematisches Ersatzmodell mit bekannter expliziter Beschreibung substituiert wird. Anhand dieses Ersatzmodells kann dann die Optimierung vorgenommen werden. Als eine universale Struktur von Ersatzmodellen eignen sich besonders die neuronalen Netzwerke, die auch zum Erlernen von Zeitreihen und den sich dahinter verbergenden physikalischen Mechanismen erfolgreich angewendet werden können. Es kann sogar gezeigt werden, dass neuronale Netzwerke mit einer hinreichenden Größe jedes dynamische System beliebig genau abbilden können.

Die Anwendung des Verfahrens besteht also im Wesentlichen aus den zwei Phasen der Systemidentifikation und der Optimierung. Dabei sind in jeder dieser Phasen verschiedene Vorgehen denkbar. So kann die Systemidentifikation wahlweise mit zeitdiskreten oder zeitstetigen neuronalen Netzen erfolgen. Die Optimierung kann ebenfalls zeitdiskret mit einem direkten Schießverfahren oder nominell zeitstetig mit einem Kollokationsverfahren angegangen werden. Jede dieser Optionen hat seine eigenen Vorteile, Fehlerquellen und möglichen Probleme.

Nach der Entwicklung und Implementierung des Gesamtverfahrens im ersten Teil dieser Arbeit wurde dieses im zweiten Teil der Arbeit auf drei unterschiedliche Testfälle angewendet, um dessen praktischen Wert zu untersuchen.

Als erstes, einfaches Anwendungsbeispiel wurde das Umschalten einer Mischungsstrecke unter Beschränkungen an die Steuerung und der thermischen Spannung in einem dickwandigen Bauteil optimiert. Hierzu wurde zunächst ein physikalisch motiviertes Ersatzmodell gebildet, das die reale Systemdynamik allerdings nicht hinreichend genau beschreiben konnte. Im Vergleich hierzu konnten sowohl mit zeitdiskreten als auch zeitstetigen neuronalen Netzen gute Ersatzmodelle entworfen werden. Bei der Optimierung stellte sich jedoch heraus, dass das zeitdiskrete NARX-Modell nicht genau genug war, sodass die Optimierungsergebnisse auf Basis der zeitstetigen CTRNN-Netzwerke wesentlich näher am realen Optimum lagen, wie in diesem einfachen Fall intuitiv überprüft werden konnte.

Im zweiten Testfall stand die Optimierung des Warmstarts einer GuD-Anlage in Malaysia im Vordergrund. Ziel war es, die Gasturbine schnellstmöglich hochzufahren, ohne dass es in der Hochdrucktrommel des nachgeschalteten Abhitzedampferzeugers zu unzulässig hohen thermischen Spannungen kommt. Außerdem waren die maximalen Lastgradienten der Gasturbine beschränkt. Zur Bildung eines Ersatzmodells konnte auf ein bestehendes APROS-Modell zurückgegriffen werden. Mit dessen Hilfe wurden Trainingsdaten gewonnen, um zeitdiskrete NARX-Netze als Ersatzmodelle zu erzeugen. Die Optimierung anhand des erzeugten Ersatzmodells wurde schließlich mit einem direkten Kollokationsverfahren vorgenommen. Die

optimalen Steuerungen wurden für zwei unterschiedliche, maximal zulässige thermische Spannungen am Originalmodell validiert und mit einer Referenztrajektorie verglichen. Hieraus ergab sich, dass der reale Warmstart der Anlage bei annähernd gleicher thermischer Schädigung um etwa 90 Minuten beschleunigt werden könnte.

Als letzter Anwendungsfall wurde der Lastwechsel des steinkohlegefeuerten Blocks 7 im Heizkraftwerk Heilbronn untersucht. Das Optimierungsziel umfasste hier eine Beschleunigung des Lastwechsels von 80 auf 100 % gegenüber einer Referenztrajektorie, wobei Einschränkungen bezüglich der Steuerungen und insbesondere der Störung der Frischdampftemperatur in das Optimierungsproblem eingeführt wurden. Auch in diesem Fall konnte auf ein bestehendes, sehr detailliertes APROS-Modell der Anlage zurückgegriffen werden, mit dessen Hilfe einerseits die Referenztrajektorie erzeugt wurde. Andererseits wurde das Modell nach kleineren Anpassungen zum Ermöglichen einer Optimierung auch dazu genutzt, Trainingsdaten zu gewinnen. Mithilfe dieser Daten wurde letztlich ein zeitdiskretes Ersatzmodell gebildet, wobei in diesem Beispiel erstmals mehrere Eingangs- und Ausgangsgrößen Teil des Problems waren. Eine Optimierung auf Basis dieses Ersatzmodells für verschiedene zulässige Abweichungen der Frischdampftemperatur ließ den Schluss zu, dass die in der realen Anlage implementierten Regelungsschaltungen einen Lastwechsel ermöglichen, der nahe am Zeitoptimum liegt. Zudem konnte ermittelt werden, dass während des Anfahrvorgangs erheblich größere Leistungen von bis zu 50 MW in das Netz eingespeist werden können, wenn höhere Schwankungsbreiten von 15 statt 3 K in der Frischdampftemperatur zugelassen werden. Die Dauer des eigentlichen Lastwechsels auf die Ziellast wird hiervon jedoch kaum beeinflusst.

Alles in Allem konnte das Verfahren in den drei Anwendungsbeispielen gute Ergebnisse liefern. Es ist jedoch festzuhalten, dass besonders in der Phase der Systemidentifikation eine Vielzahl möglicher Probleme auftreten kann. Zu deren Beseitigung existieren zwar weitreichende Mittel, deren Anwendung macht das Erzeugen der Ersatzmodelle jedoch aufwändig. Andererseits kann ein Optimierungsergebnis nur dann von Wert sein, wenn das zugrunde gelegte Ersatzmodell auch eine gute Qualität aufweist, sodass dem Schritt der Systemidentifikation höchste Bedeutung beizumessen ist.

Zum Abschluss dieser Arbeit soll im nächsten Kapitel noch ein Ausblick über mögliche Ansatzpunkte zukünftiger Arbeiten und Weiterentwicklungen des im Rahmen dieser Arbeit entwickelten Verfahrens gegeben werden.

10 Ausblick

Im Rahmen dieser Arbeit wurde versucht, einen möglichst breiten und dennoch umfassenden Überblick über alle Ebenen und Aspekte des entwickelten Verfahrens zu geben. Aufgrund der damit verbundenen thematischen Vielfalt musste an mancher Stelle auf die Darlegung von Feinheiten verzichtet werden. Dementsprechend gibt es eine Reihe möglicher Ansätze, das Optimierungsverfahren in zukünftigen Arbeiten weiterzuentwickeln.

Auf der Ebene der Systemidentifikation ist in erster Linie das Trainingsverfahren für zeitstetige CTRNN-Netze als mögliches Verbesserungspotenzial festzuhalten. Während aufgrund der nicht nur in dieser Arbeit erzielten Ergebnisse an der grundlegenden algorithmischen Idee des Trainingsverfahrens festzuhalten ist, so kann dessen Implementierung hinsichtlich der Effizienz und der erzielbaren Genauigkeit noch verbessert werden. Ein vielversprechender Ansatz hierfür wurde bereits in der Literatur verfolgt (Al Seyab 2006, Cao 2005, Cao u. Al Seyab 2006). So konnte durch die Anwendung von Taylorreihenentwicklungen der am Trainingsverfahren beteiligten Funktionen in Verbindung mit automatischer Differentiation einerseits die Dauer des Trainingsprozesses stark reduziert werden. Andererseits wurden auf diese Weise durch eine einfache Erhöhung der Ordnung der Taylorreihenentwicklungen bessere Trainingsergebnisse erzielt als mit dem in dieser Arbeit verwendeten Lernverfahren, das auf der numerischen Lösung der Sensitivitätsgleichungen basiert (Al Seyab 2006).

Auch von Seiten der eigentlichen Optimierung lässt sich Verbesserungspotenzial festhalten. Eine mögliche Erweiterung des Gesamtverfahrens umfasst die nachträgliche Schätzung der adjungierten Variablen $\lambda(t)$ und $\eta(t)$ in Gleichung (3.23) aus dem Optimierungsergebnis. Diese enthalten Informationen, die einerseits die Überprüfung der tatsächlichen Optimalität des Ergebnisses erlauben. Andererseits können sie zur Beurteilung möglicher weiterer Reduktionen des Kostenfunktionals bei der Lockerung der Beschränkungen herangezogen werden. Dieses Vorgehen ist allerdings nur in Verbindung mit einem direkten Kollokationsverfahren möglich (Stryk 1995, Stryk u. Bulirsch 1992, Kirk 1970).

Darüber hinaus sind Weiterentwicklungen des direkten Schießverfahrens vielversprechend, da hierdurch insbesondere auf eine approximierende Umwandlung eines NARX-Netzes in ein zeitstetiges Ersatzmodell verzichtet werden kann. Besonders lohnend ist hier die Integration der bekannten SQP-Implementierung SNOPT, welche derzeit als einer der effizientesten Algorithmen zur Lösung von statischen, nichtlinearen Optimierungsproblemen gilt (Gill u. a. 2008). Denkbar wäre auch, dem Optimierungsalgorithmus exakte Ableitungsinformationen zur Verfügung zu stellen, sodass dieser die Ableitungen nicht mehr mittels finiter Differenzen approximieren muss. Dies hätte den Vorteil einer höheren Genauigkeit und einer schnelleren Konvergenz des Optimierungsverfahrens (Papageorgiou 2006, Kelley 1999). Zur Ermittlung der Ableitungen würde sich die automatische Differentiation anbieten, die auf die neuronalen Netzwerke anzuwenden wäre.

Zuletzt sei noch ein sehr wichtiger Ansatzpunkt zukünftiger Arbeiten angemerkt, der in der vorliegenden Arbeit nur peripher diskutiert wurde. Das Ergebnis des Gesamtverfahrens in seiner jetzigen Form ist die optimale Steuerung eines Prozesses. Wird diese nun in den realen Prozess eingespeist, so ist es sehr wahrscheinlich, dass es zu Abweichungen der tatsächlichen Trajektorie gegenüber dem Optimierungsergebnis kommt. Diese sind auf Modellungenauigkeiten und Störungen im Prozess zurückzuführen, welche niemals zu vollständig auszuschließen sind. Aus diesem Grunde wäre, anstatt einer optimalen *Steuerung* $u(t)$, eine optimale *Regelung* $u(x)$ wünschenswert, die den tatsächlichen Zustand des realen Prozesses in die Ermittlung des Stelleingriffs mit einbezieht. Die Berechnung eines derartigen Regelgesetzes ist leider

nur in sehr wenigen, einfachen Einzelfällen mit vertretbarem Aufwand machbar oder überhaupt möglich. Ein prominentes Beispiel für derartige Einzelfälle sind rein lineare Systeme mit quadratischen Gütekriterien bei fester Endzeit und ohne Beschränkungen (Papageorgiou 2006). Eine derartige Einschränkung ist für das vorliegende Verfahren nicht akzeptabel, da es insbesondere die Forderung nach Flexibilität, die in Abschnitt 2.1 formuliert wurde, verletzt.

Um dennoch die Probleme im Umgang mit der optimalen Steuerung zu vermeiden, existieren einige Techniken. So ist beispielsweise eine einfache Sollwerttrajektorien-Folgeregelung denkbar, bei der nicht die eigentliche optimale Steuerung im realen Prozess verwendet wird, sondern die optimalen Zustandstrajektorien als Führungsgröße für Regler dienen, die den Prozess so nah wie möglich an den optimalen Prozess heranführen sollen. Hierdurch geht allerdings die Optimalität verloren. Zudem könnten durch den Regeleingriff Steuerungsbeschränkungen verletzt werden.

Darüber hinaus ließe sich eine sogenannte modellprädiktive Regelung anwenden. Hierbei handelt es sich um eine dynamische Neu-Optimierung der Steuerungen auf einem festen Zeithorizont zur Kompensation von Störungen und Modellungenauigkeiten (Papageorgiou 2006, Milam 2003, Al Seyab 2006). Derartige Techniken sind teilweise Stand der Technik und wurden auch schon in Verbindung mit zeitstetigen neuronalen Netzen angewendet (Al Seyab 2006, Cao 2005, Cao u. Al Seyab 2006).

Eine weiteres Verfahren zur Vermeidung der genannten Probleme ist die Methode der benachbarten Extremalen. Diese umfasst die numerische Synthese der optimalen Regelung aus vielen einzelnen optimalen Steuerungen (Kim u. Hill 1994, Jiang u. a. 2012).

Literaturverzeichnis

Ahn u. Song 2013
Ahn, Choon K. ; Song, Moon K.: Peak-to-peak exponential direct learning of continuous-time recurrent neural network models: a matrix inequality approach. In: *Journal of Inequalities and Applications* 1 (2013), Nr. 68

Akesson 2007
Akesson, Johan: *Languages and Tools for Optimization of Large-Scale Systems.* Lund, Lund University, Diss., 2007

Akesson 2008
Akesson, Johan: Optimica - An Extension of Modelica Supporting Dynamic Optimization. In: Modelica Association (Hrsg.): *6th International Modelica Conference 2008.* 2008

Akesson u. a. 2009
Akesson, Johan ; Gäfvert, M. ; Tummescheit, H.: JModelica - an Open Source Platform for Optimization of Modelica Models. In: TU Wien (Hrsg.): *Proceedings of MATHMOD 2009 - 6th Vienna International Conference on Mathematical Modelling.* Wien, 2009

Al Seyab 2006
Al Seyab, Rihab: *Nonlinear Model Predictive Control Using Automatic Differentiation.* Cranfield, University of Cranfield, Diss., 2006

Al Seyab u. Cao 2008
Al Seyab, R.K ; Cao, Y.: Nonlinear system identification for predictive control using continuous time recurrent neural networks and automatic differentiation. In: *Journal of Process Control* 18 (2008), Nr. 6, S. 568–581

Al-Zaharnah 2002
Al-Zaharnah, Iyad: *Thermal Stresses in Pipes.* Dublin and Ireland, Dublin City University, Diss., 2002

Albanesi u. a. 2006
Albanesi, Chiara ; Bossi, Marco ; Magni, Lalo ; Paderno, Jurij ; Pretolani, Francesco ; Peter Kühl ; Diehl, Moritz: Optimization of the Start-Up Procedure of a Combined Cycle Power Plant. In: Misra, Pradeep (Hrsg.) ; Middleton, Richard H. (Hrsg.): *45th IEEE Conference on Decision and Control*, Institute of Electrical and Electronics Engineers, 2006. – ISBN 9781424401710, S. 1840–1845

Ali u. Schmid 2010
Ali, Abid ; Schmid, Christian: System Identification Using Neural Networks. In: *Control Systems, Robotics and automation* 6 (2010), S. 1

Alobaid u. a. 2014a
Alobaid, Falah ; Starkloff, Ralf ; Pfeiffer, Stefan ; Karner, Karl ; Epple, Bernd ; Kim, Hyun-Gee: A comparative study of different dynamic process simulation codes for combined cycle power plants: part A: During part loads and off-design operation. In: *Fuel Journal* (2014)

Alobaid u. a. 2014b
Alobaid, Falah ; Starkloff, Ralf ; Pfeiffer, Stefan ; Karner, Karl ; Epple, Bernd ; Kim, Hyun-Gee: A comparative study of different dynamic process simulation codes for combined cycle power plants: part B: During start-up procedure. In: *Fuel Journal* (2014)

Atuonwu u. a. 2010
Atuonwu, J.C ; Cao, Y. ; Rangaiah, G.P ; Tade, M.O: Identification and Predictive Control of a Multistage Evaporator. In: *Control Engineering Practice* 18 (2010), Nr. 12, S. 1418–1428

Beer 1995
Beer, Randall D.: On the Dynamics of Small Continuous-Time Recurrent Neural Networks. In: *Adaptive Behavior* 3 (1995), Nr. 4, S. 469–509

Bellman 1961
Bellman, Richard: *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 1961

Bischof u. a. 2003
Bischof, Christian ; Lang, Bruno ; Vehreschild, Andre: Automatic Differentiation for MATLAB Programs. In: *PAMM* 2 (2003), Nr. 1, S. 50–53

Bourdache-Siguerdidjane u. Fliess 1987
Bourdache-Siguerdidjane, Houria ; Fliess, Michel: Optimal feedback control of non-linear systems. In: *Automatica* 23 (1987), Nr. 3, S. 365–372

Bräuer u. Gericke 2014
Bräuer, Ralf ; Gericke, Bernd: Zwangdurchlauf-Dampferzeuger in GuD-Anlagen für dynamischen Netzbetrieb. In: *VGB Powertech* (2014), Nr. 7, S. 58–65

Cao 2005
Cao, Yi: A formulation of nonlinear model predictive control using automatic differentiation. In: *Journal of Process Control* 15 (2005), Nr. 8, S. 851–858

Cao u. Al Seyab 2006
Cao, Yi ; Al Seyab, Rihab: Differential Recurrent Neural Network based Predictive Control. In: Marquardt, W. (Hrsg.) ; Pantelides, C. (Hrsg.): *16th European Symposium on Computer Aided Process Engineering and 9th International Symposium on Process Systems Engineering* Bd. 21A-21B. Amsterdam : Elsevier, 2006. – ISBN 9780444529695, S. 1239–1244

Casella u. a. 2011a
Casella, Francesco ; Donida, Filippo ; Akesson, Johan: Object-Oriented Modeling and Optimal Control: A Case Study in Power Plant Start-Up. In: Bittanti, Sergio (Hrsg.): *IFAC 2011* Bd. 18, IFAC, 2011 (World congress part 1). – ISBN 3902661933, S. 9549–9554

Casella u. a. 2011b
Casella, Francesco ; Farina, Marcello ; Righetti, Fabio ; Scattolini, Riccardo ; Faille, Damien ; Davelaar, Frans ; Tica, Adrian ; Gueguen, Herve ; Dumur, Didier: An Optimization Procedure of the Start-Up of Combined Cycle Power Plants. In: Bittanti, Sergio (Hrsg.): *IFAC 2011* Bd. 18, IFAC, 2011 (World congress part 1). – ISBN 3902661933, S. 7043–7048

Chen u. Wermter 1998
Chen, Joseph ; Wermter, Stefan: *Continuous Time Recurrent Neural Networks for Grammatical Induction*. Hamburg, 1998

Chow u. Li 2000
Chow, Tommy W. ; Li, Xiao-Dong: Modeling of continuous time dynamical systems with input by recurrent neural networks. In: *IEEE Transactions on Circuits and Systems* 47 (2000), Nr. 4, S. 575–578

Cybenko 1989
Cybenko, G.: Approximation by superpositions of a sigmoidal function. In: *Math. Control Signals Systems* 3 (1989), Nr. 2, S. 303–314

Delgado u. a. 1995
Delgado, A. ; Kambhampati, C. ; Warwick, K.: Dynamic recurrent neural network for system identification and control - Control Theory and Applications, IEE Proceedings-. In: *IEEE Process Control Theory and Application* 142 (1995), Nr. 4, S. 307–314

Demuth u. a. 2009
Demuth, Howard ; Beale, Mark ; Hagan, Martin T.: *Matlab Neural Network Toolbox 6: User's Guide*. 2009

Diaconescu 2008
Diaconescu, Eugen: The use of NARX Neural Networks to predict Chaotic Time Series. In: *WSEAS Transactions on Computer Research* 3 (2008), Nr. 3, S. 182–191

Dolezal 2001
Dolezal, Richard: *Kombinierte Gas- und Dampfkraftwerke*. Berlin : Springer-Verlag, 2001

Doya 1993
Doya, Kenji: Universality of Fully-Connected Recurrent Neural Networks. In: *IEEE Transactions on Neural Networks* (1993)

EnBW Kraftwerke AG 2010
EnBW Kraftwerke AG: *Das Heizkraftwerk Heilbronn und seine dezentralen Standorte Walheim und Marbach*. Version: 2010. `https://www.enbw.com/media/konzern/docs/energieerzeugung/enbw-flyer_heizkraftwerk_heilbronn.pdf`

Epple u. a. 2012
Epple, Bernd ; Leithner, Reinhard ; Linzer, Wladimir ; Walter, Heimo: *Simulation von Kraftwerken und Feuerungen*. 2. Wien : Springer, 2012. – ISBN 370911182X

Falco u. a. 2008
Falco, Ivanoe d. ; Della Cioppa, Antonio ; Donnarumma, Francesco ; Maisto, Domenico ; Prevete, Roberto ; Tarantino, Ernesto: CTRNN parameter learning using Differential Evolution. In: Ghallab, Malik (Hrsg.): *ECAI 2008* Bd. 178. Amsterdam and Oxford : IOS Press, 2008. – ISBN 978–1–58603–891–5

Funahashi u. Nakamura 1993
Funahashi, Ken-Ichi ; Nakamura, Yuichi: Approximation of dynamical systems by continuous time recurrent neural networks. In: *Neural Networks* 6 (1993), Nr. 6, S. 801–806

Gallagher u. Vigraham 2002
Gallagher, John ; Vigraham, Saranyan: *A Modified Compact Genetic Algorithm for the Intrinsic Evolution of Continuous Time Recurrent Neural Networks*. Dayton, 2002

Georgakis 2012
Georgakis, Christos: A Model-Free Methodology for the Optimization of Batch Processes: Design of Dynamic Experiments. In: IFAC (Hrsg.): *8th IFAC International Symposium on Advanced Control of Chemical Processes*. Red Hook and NY : Curran, 2012. – ISBN 9781622762286

Gill u. a. 2008
Gill, Philip E. ; Murray, Walter ; Saunders, Michael A.: *User's Guide for SNOPT Version 7: Software for Large-Scale Nonlinear Programming.* 2008

Grammenoudis u. Weber 2011
Grammenoudis, Paschalis ; Weber, Jochen: *Aktuelle Sicht der Lebensdauerüberwachung der HT-/HD-Rohrleitungssysteme in konventionellen Kraftwerken.* Stuttgart, 2011 (26. FDBR-Fachtagung Rohrleitungstechnik)

Hagan u. Menhaj 1994
Hagan, Martin T. ; Menhaj, Mohammad B.: Training feedforward networks with the Marquardt algorithm. In: *IEEE Transactions on Neural Networks* 5 (1994), Nr. 6, S. 989–993

Haykin 1999
Haykin, Simon S.: *Neural networks: A comprehensive foundation.* 2. Upper Saddle River and N.J : Prentice Hall, 1999. – ISBN 0–13–908385–5

He u. Asada 1993
He, Xiangdong ; Asada, Haruhiko: A New Method for Identifying Orders of Input-Output-Models for Nonlinear Dynamic Systems. In: IEEE (Hrsg.): *American Control Conference, 1993*, IEEE, 1993. – ISBN 0780308603, S. 2520–2523

Heinzel u. a. 2012
Heinzel, Timm ; Meiser, Albrecht ; Stamatelopoulos, Georg-Nikolaus ; Buck, Peter: Einführung Einmühlenbetrieb in den Kraftwerken Bexbach und Heilbronn Block 7. In: *VGB Powertech* (2012), Nr. 11, S. 85–89

Hemami u. a. 1992
Hemami, A. ; Mehrabi, M.G ; Cheng, R.M.H: Synthesis of an Optimal Control Law for Path Tracking in Mobile Robots. In: *Automatica* 28 (1992), Nr. 2, S. 383–387

Horvath 2003
Horvath, Gabor: *Neural Networks in System Identification.* Budapest, 2003

Jiang u. a. 2012
Jiang, Canghua ; Teo, Kok L. ; Loxton, Ryan ; Duan, Guang-Ren: A neighboring extremal solution for optimal switched impulsive control problems with large perturbations. In: *International Journal of Innovative Computing, Information and Control* 8 (2012), Nr. 9, S. 6235–6257

Kambhampati u. a. 2000
Kambhampati, C. ; Garces, F. ; Warwick, K.: Approximation of non-autonomous dynamic systems by continuous time recurrent neural networks - Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conferenc. In: IEEE (Hrsg.): *Proceedings of the International Joint Conference on Neural Networks 2000.* 2000. – ISBN 0–7695–0619–4

Kelley 1999
Kelley, C. T.: *Iterative methods for optimization.* Philadelphia : SIAM, 1999 (Frontiers in applied mathematics). – ISBN 9780898714333

Kim u. a. 2004
Kim, Il-Hwan ; Fok, Stanley ; Kingsley, Fregene ; Lee, Dong-Hoon ; Oh, Tae-Seok ; Wang, David W.: Neural Network-Based System Identification and Controller Synthesis for an Industrial Sewing Machine. In: *International Journal of Control, Automation, and Systems.* 2 (2004), Nr. 1, S. 83

Kim u. Hill 1994
KIM, Theodore J. ; HILL, David G.: *Neighboring extremal control design including model mismatch errors.* 1994

Kirk 1970
KIRK, Donald E.: *Optimal control theory: An introduction.* Englewood Cliffs and N.J : Prentice-Hall, 1970 (Prentice-Hall networks series). – ISBN 9780486434841

Krüger u. a. 2001
KRÜGER, Klaus ; RODE, Manfred ; FRANKE, Rüdier: Optimal control for fast boiler start-up based on a nonlinear model and considering the thermal stress on thick-walled components. In: IEEE CONTROL SYSTEMS SOCIETY (Hrsg.): *Proceedings of the 2001 IEEE International Conference on Control Applications*, 2001. – ISBN 9780780367333, S. 570–576

Lampton 1997
LAMPTON, Michael: Damping–undamping strategies for the Levenberg–Marquardt nonlinear least-squares method. In: *Computers in Physics* 11 (1997), Nr. 1, S. 110–115

Lausterer 1997
LAUSTERER, G.K: On-line thermal stress monitoring using mathematical models. In: *Control Engineering Practice* 5 (1997), Nr. 1, S. 85–90

Li u. a. 2005
LI, Xiao-Dong ; HO, John K. ; CHOW, Tommy W.: Approximation of dynamical time-variant systems by continuous-time recurrent neural networks. In: *IEEE Transactions on Circuits and Systems* 52 (2005), Nr. 10, S. 656–660

Lind u. Sällberg 2012
LIND, Alexandra ; SÄLLBERG, Elin: *Optimization of the Start-up Procedure of a Combined Cycle Power Plant.* Lund, Lund University, Diss., 2012

Magnusson 2012
MAGNUSSON, Fredrik: *Collocation methods in JModelica.org.* Lund, Lund University, Diss., 2012

Marquardt 1963
MARQUARDT, Doland W.: An Algorithm for Least-Squares Estimation of Nonlinear Parameters. In: *Journal of the Society for Industrial and Applied Mathematics* 11 (1963), Nr. 2, S. 431–441

Milam 2003
MILAM, Mark B.: *Real-Time Optimal Trajectory Generation for Constrained Dynamical Systems.* Pasadena and California, California Institute of Technology, Diss., 2003

Mitchell 1996
MITCHELL, Melanie: *An introduction to genetic algorithms.* Cambridge : MIT Press, 1996. – ISBN 0–262–13316–4

Modelon AB 2014
MODELON AB: *JModelica User Guide: Version 1.14.* `http://www.jmodelica.org/api-docs/usersguide/JModelicaUsersGuide-1.14.0.pdf`. Version: 2014

Moré 1978
MORÉ, Jorge J.: The Levenberg-Marquardt algorithm: Implementation and theory. In: WATSON, G. A. (Hrsg.): *Numerical analysis* Bd. 630. Berlin and New York : Springer-Verlag, 1978. – ISBN 9783540085386

Nakamura u. Nakagawa 2009
NAKAMURA, Yuichi ; NAKAGAWA, Masahiro: Approximation Capability of Continuous Time Recurrent Neural Networks for Non-autonomous Dynamical Systems. In: ALIPPI, Cesare (Hrsg.): *Artificial neural networks* Bd. 5769. Berlin [u.a.] : Springer, 2009. – ISBN 3–642–04276–7, S. 593–602

Nelder u. Mead 1965
NELDER, J. A. ; MEAD, R.: A Simplex Method for Function Minimization. In: *Computer Journal* 7 (1965), Nr. 1, S. 308–313

Nielsen 1999
NIELSEN, Hans B.: *Damping Parameter in Marquardt's Method*. Lyngby, 1999

Nocedal u. Wright 1999
NOCEDAL, Jorge ; WRIGHT, Stephen J.: *Numerical optimization*. New York : Springer, 1999 (Springer series in operations research). – ISBN 9780387227429

Norgaard 1997
NORGAARD, Magnus ; TECHNICAL UNIVERSITY OF DENMARK (Hrsg.): *NNSYSID: Neural Network Based System Identification Toolbox*. `http://www.iau.dtu.dk/research/control/nnsysid.html`. Version: 1997

Papageorgiou 2006
PAPAGEORGIOU, M.: *Optimierung: Statische, dynamische, stochastische Verfahren*. 3. Berlin : Springer, 2006. – ISBN 978–3–540–34012–6

Papula 2008
PAPULA, Lothar: *Mathematik für Ingenieure und Naturwissenschaftler Band 3: Vektoranalysis, Wahrscheinlichkeitsrechnung, Mathematische Statistik, Fehler- und Ausgleichsrechnung*. 5. Wiesbaden : Vieweg + Teubner Verlag, 2008. – ISBN 978–3–8348–0225–5

Pinho u. Vinter 1997
PINHO, M. d. ; VINTER, R.B: Necessary Conditions for Optimal Control Problems Involving Nonlinear Differential Algebraic Equations. In: *Journal of Mathematical Analysis and Applications* (1997), Nr. 212, S. 493–516

Potter 2006
POTTER, Richard: *An Investigation into the Dynamics of a Continuous Time Recurrent Neural Network Node*. Sussex, 2006

Ross u. a. 2008
ROSS, I. M. ; SEKHAVAT, Pooya ; FLEMING, Andrew ; GONG, Qi: Optimal Feedback Control: Foundations, Examples, and Experimental Results for a New Approach. In: *Journal of Guidance, Control, and Dynamics* 31 (2008), Nr. 2, S. 307–321

Schäfer u. Zimmermann 2006
SCHÄFER, Anton M. ; ZIMMERMANN, Hans G.: Recurrent Neural Networks Are Universal Approximators. In: KOLLIAS, Stefanos (Hrsg.): *Artificial neural networks* Bd. 4131-4132, Springer, 2006 (LNCS sublibrary. SL 1, Theoretical computer science and general issues). – ISBN 3540388737, S. 632–640

Scheibner 2014
SCHEIBNER, Gunter: Die Energiewende und ihre Herausforderungen an die Systemführung. In: *VGB Powertech* (2014), Nr. 6, S. 38–42

Schmidt u. Schuele 2013
SCHMIDT, Gerald ; SCHUELE, Volker: *Anpassung thermischer Kraftwerke an künftige Herausforderungen im Strommarkt.* Berlin, 2013

Sola u. Sevilla 1997
SOLA, J. ; SEVILLA, J.: *Importance of Input Data Normalization for the Application of Neural Networks to Complex Industrial Problems.* Pamplona and Spain, 1997

Sragner u. Horvath 2003
SRAGNER, László ; HORVATH, Gabor: Improved Model Order Estimation for Nonlinear Dynamic Systems. In: *International Scientific Journal of Computing* 2 (2003), Nr. 2, S. 93–100

Starkloff u. a. 2013
STARKLOFF, Ralf ; KARNER, Karl ; EPPLE, Bernd ; BÖHM, Felix ; STAMATELOPOULOS, Georg-Nikolaus: Erstellung und Validierung eines Modells zur Untersuchung transienter Vorgänge in konventionellen Kraftwerken hinsichtlich Flexibilität. In: BECKMANN, Michael (Hrsg.) ; HURTADO, Antonio (Hrsg.): *Kraftwerkstechnik.* Neuruppin : TK, 2013. – ISBN 978–3–944310–05–3, S. 53–62

Stryk 1993
STRYK, O. v.: Numerical Solution of Optimal Control Problems by Direct Collocation. In: BULIRSCH, R. (Hrsg.) ; MIELE, A. (Hrsg.) ; STOER, J. (Hrsg.) ; WELL, K. (Hrsg.): *Optimal Control* Bd. 111. Basel : Birkhäuser Basel, 1993. – ISBN 3034875398, S. 129–143

Stryk 1995
STRYK, O. v.: *Numerische Lösung optimaler Steuerungsprobleme: Diskretisierung, Parameteroptimierung und Berechnung der adjungierten Variablen.* München, Technische Universität München, Diss., 1995

Stryk u. Bulirsch 1992
STRYK, O. v. ; BULIRSCH, R.: Direct and indirect methods for trajectory optimization. In: *Annals of Operations Research* (1992), Nr. 32, S. 357–373

The MathWorks 2004
THE MATHWORKS: *Genetic Algorithm and Direct Search Toolbox™ 2.* 2004

Transtrum u. Sethna 2012
TRANSTRUM, Mark K. ; SETHNA, James P.: Improvements to the Levenberg-Marquardt algorithm for nonlinear least-squares minimization. In: *Journal of Computational Physics* (2012)

VDI 2006
VDI: *VDI-Wärmeatlas: Berechnungsunterlagen für Druckverlust, Wärme- und Stoffübertragung.* 10. Berlin : Springer, 2006. – ISBN 3–540–25504–4

Wächter u. Bieger 2006
WÄCHTER, Andreas ; BIEGER, Lorenz T.: On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. In: *Mathematical Programming* 106 (2006), Nr. 106, S. 25–57

Zehtner 2009
ZEHTNER, Wolfgang F.: *Betriebsoptimierung von Steinkohlekraftwerken durch Simulation.* 1. München : Verl. Dr. Hut, 2009 (Energietechnik). – ISBN 978–3–86853–239–5

A Anhang

A.1 Quellcodes zur Systemidentifikation

Programmcode A.1: Matlab-Code zur Lipschitz-Analyse.

```
clear all
addpath 'NNSYSID_Toolbox'

sim_data=csvread('trainingdata.csv');
sim_y = sim1(:,3);
sim_u = sim1(:,2);

OrderIndices = lipschit(sim_u,sim_y,1:10,1:10)
```

Programmcode A.2: Matlab-Code zum Training von NARX-Netzwerken.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% learn.m                                                               %
% Neuronale Netze mit fester Komplexität mehrfach trainieren und        %
% manuelle Auswahl des besten Ergebnisses                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all;

% nötige Einstellungen zum Training:
nTrials = 30;           % Anzahl der Trainingswiederholungen
inputDelays = 1:2;      % Verzögerungsordnung für u
feedbackDelays = 1:2;   % Verzögerungsordnung für y
hiddenLayerSize = 50;   % Anzahl der Neuronen in der versteckten Schicht

% Trainingsdaten lesen und aufbereiten
tmp=csvread('dymola.csv');
sim_time = tmp(:,1);
sim_u = tmp(:,3);
sim_y = tmp(:,2);
sim_y=1e-6 * sim_y;

inputSeries = tonndata(sim_u,false,false);
targetSeries = tonndata(sim_y,false,false);

nets=cell(nTrials,2);   %Spalten: 1. net-Objekte, 2. error-Struct

for i=1:nTrials

    disp(['Performing trial number ' num2str(i) '...']);

    % Netz erzeugen
    net2 = narxnet(inputDelays,feedbackDelays,hiddenLayerSize);
    net2.divideFcn = 'divideblock';
    net2.divideParam.trainRatio = 75/100;
    net2.divideParam.valRatio = 25/100;
    net2.divideParam.testRatio = 0/100;
    net2.trainFcn = 'trainlm';

    % Netz im offenen Kreis trainieren
    [Xs,Xi,Ai,Ts] = preparets(net2,inputSeries,{},targetSeries);
    net2.trainParam.showWindow = false;
    net2.trainParam.showCommandLine = true;
    net2.trainParam.epochs = 1000;
    net2.trainParam.show = 100;

    [net2,tr] = train(net2,Xs,Ts,Xi,Ai,'useParallel','yes');
    Y = net2(Xs,Xi,Ai);

    % Kreis des Netzes schließen und Netzperformance auswerten
    net_closeloop = closeloop(net2);
    [Xs,Xi,Ai,Ts] = preparets(net_closeloop,inputSeries,{},targetSeries);
    Y = net_closeloop(Xs,Xi,Ai);

    tmp2 = cell2mat(gsubtract(Ts,Y));
    tmp=corrcoef(cell2mat(Ts),cell2mat(Y));
    err=struct('predictionError',tmp2,'maxAbsError',max(abs(tmp2)),'meanAbsError',
        mean(abs(tmp2)),'stdAbsError',std(abs(tmp2)),'correlation',tmp(1,2));

    disp(['Done performing trial number ' num2str(i) '. Maximum error: ' num2str(err.
        maxAbsError) '. Correlation: ' num2str(err.correlation)]);
```

```
58
59      % Performance-Kennwerte abspeichern
60      errs(i) = err.maxAbsError;
61      corrs(i) = err.correlation;
62
63      nets{i,1}=net2;
64      nets{i,2}=err;
65
66  end
67
68  % Korrelationswert und maximale Abweichung jedes Netzes anzeigen
69  figure
70  subplot(2,1,1)
71  plot(errs)
72  subplot(2,1,2)
73  plot(corrs,'r')
```

Programmcode A.3: Matlab-Code zur gradientenbasierten Training von CTRNN-Netzen der Form (3.7a) und (3.7b).

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% train.m                                                                %
% zeitstetige Neuronale Netze mit fester Komplexität mehrfach    %
% trainieren                                                             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all

% Trainingsdaten einlesen un aufbereiten
tmp=csvread('C:\Users\Robert\Desktop\Matlab_NN\NN-Test mit sigma\new_0_1\dymola_val.
    csv');
for m=1:25 % (dymola-bedingte) Doppelwerte löschen
    for i=1+1:size(tmp,1)
        if tmp(i,1)==tmp(i-1,1)
            tmp(i,:)=[];
            break
        end
    end
end

sim_time = tmp(:,1);
sim_u = tmp(:,3);
sim_y = 1e-9*tmp(:,2);

Tspan = sim_time;
u(1,:) = sim_u';
ut=sim_time';
X=sim_y;

% Ableitung von u numerisch bestimmen und als zweite Steuerung übergeben
dudt=diff(u(1,:))./diff(ut);
u(2,:) = smooth([0 dudt],11)';

% Spezialfall: Hier nur Ableitung von u im Trainingsprozess berücksichtigen
u = u(2,:);

% Verfahrensparameter setzen
n_x=1; % Anzahl der internen Zustände
n_u=1; % Anzahl der Steuerungen
n_H=1; % Anzahl der Neuronen in der versteckten Schicht
n_y=1; % Anzahl der Ausgangsgrößen

n_eta=n_H*n_x   +    n_H*n_x   +   n_H*n_u   +    n_H   +   n_x;

% Anfangsbedingungen
IC = zeros((1+n_eta)*n_x,1); % y(t=0) = 1
IC(1:n_x) = 0;

% Anfänglicher Dämpfungsfaktor
mu=1000;

% Verfahren initialisieren

E=zeros(size(ut,2),n_y);
J=zeros(size(ut,2)*n_y,n_eta);
J_tmp=zeros(n_eta,size(ut,2)*n_y);
e = zeros(n_y ,size(ut,2));

```

```
errold = 1e90;

eta = 1*(-1 + 2*rand(n_eta,1));

calcSens = 1;

% Verfahren starten

C= [eye(n_y), zeros(n_y, n_x-n_y)];
etatmp=eta;

for k=1:2500 % Mehrere Lern-Epochen (Iterationen)

    % Aktuelle Parameterschätzung auswerten
    options = odeset('RelTol',1e-4,'AbsTol',1e-4,'MaxStep',7.5);
    [T, Xtrain] = ode15s(@(t,x) ode_rhs(t,x,ut,u,etatmp,n_x,n_u,n_H,calcSens),Tspan,
        IC,options); % Solve ODE
    Y = C * Xtrain(:,1:n_x)';

    for i= 1:size(ut,2)
        e(:,i)=transpose(Y(i)-X(i,:));
        J_tmp(:,(i-1)*n_y+1:i*n_y) = C*(reshape(Xtrain(i,n_x+1:(1+n_eta)*n_x),n_x,
            n_eta));
    end

    E=reshape(e,[size(Tspan,1)*n_y,1]);
    J=J_tmp';

    err=   norm(0.5*transpose(E)*E);

    % Performance auswerten
    corr=corrcoef(X,Y);
    disp([num2str(k) '. Iteration.' 9 9 9 'Mu: ' num2str(mu) 9 9 9 'This Error: '
        num2str(err) 9 9 9 'Old Error: ' num2str(errold) 9 9 9 'Corr: ' num2str(corr
        (2,1)) 9 9 9 'MaxErr: ' num2str(max(abs(E))) ])


    %Kandidaten für nächste Iteration ermitteln
    delta_eta = - ( transpose(J)*J + mu*eye(n_eta)) \ transpose(J)*E;
    eta = etatmp;
    etatmp=eta + delta_eta;

    plot(T, X); hold on;plot(T,Xtrain(:,1:1),'--r');hold off
    drawnow

end

% Ergebnisse anzeigen
plot(T, X); hold on;plot(T,Xtrain(:,1:1),'--r');hold off
title('Plot of y as a function of time');
xlabel('Time'); ylabel('Y(t)');
```

Programmcode A.4: Matlab-Code zum gradientenfreien Training von CTRNN-Netzen der Form (3.7a) und (3.7b).

```
function [p_estimate,fval,exitflag,output,iterhistory] = train_gradfrei
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% train_gradfrei.m                                                      %
% Trainieren eines CTRNN-Netzes mit verschiedenen gradienten-    %
% freien Algorithmen von Matlab                                         %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all

% Trainingsdaten einlesen und aufbereiten
tmp=csvread('C:\Users\Robert\Desktop\Matlab_NN\NN-Test mit sigma\new_0_1\dymola_val.
    csv');
for m=1:25
    for i=1+1:size(tmp,1)
        if tmp(i,1)==tmp(i-1,1)
            tmp(i,:)=[];
            break
        end
    end
end

sim_time = tmp(:,1);
sim_u = tmp(:,3);
sim_y = 1e-9*tmp(:,2);

Tspan = sim_time;
u(1,:) = sim_u';
ut=sim_time';
X=sim_y;

% Ableitung der Steuerung berechnen
dudt=diff(u(1,:))./diff(ut);
u(2,:) = smooth([0 dudt],11)';

% Spezialfall hier: Nur du/dt mit einbeziehen
u = u(2,:);


% Verfahrensparameter setzen
method = 'genetic';    % 'patternsearch', 'neldermead', 'levenbergmarquardt', '
    genetic'
CTRNNForm = 'classical';   % 'classical', 'alseyab'

n_x=2; % Anzahl der internen Zustände
n_u=1; % Anzahl der Steuerungen
n_H=1; % Anzahl der versteckten Neuronen (nur für seyab-Form)
n_y=1; % Anzahl der Ausgangsgrößen

if strcmp(CTRNNForm,'classical')
    n_eta=n_x   +    n_x^2   +    n_x;
elseif strcmp(CTRNNForm,'alseyab')
    n_eta=n_H*n_x   +    n_H*n_x   +   n_H*n_u   +    n_H   +   n_x; %alseyab
end

biaszero = true; % Sollen die bias-Gewichte des Netzes = 0 gesetzt werden?

IC = zeros((1+n_eta)*n_x,1); % y(t=0) = 1
IC(1:n_x) = 0;
```

```
56
57
58  % Verfahren initialisieren
59  eta = 1*(-1 + 1*1*rand(n_eta,1));
60
61  calcSens = 0;
62
63  iterhistory.param = [];
64  iterhistory.fval = [];
65  iterhistory.funccount = [];
66
67
68  % Verfahren starten
69  C= [eye(n_y), zeros(n_y, n_x-n_y)];
70  etatmp=eta;
71
72  if biaszero == true
73      etatmp=etatmp(1:n_eta-n_x);
74  end
75
76  % Das jeweilige Verfahren aufrufen
77  switch method
78
79      case 'neldermead'
80
81          opt = optimset('Display',       'iter',...
82                         'PlotFcns',      {@optimplotx @optimplotfval},...
83                         'OutputFcn',     {@fmincon_outfun},...
84                         'TolFun',        1e-8,...
85                         'TolX',          1e-8);
86          [p_estimate,fval,exitflag,output] = fminsearch(@(p)odefit(ut,u,p,X),etatmp,
                opt);
87
88      case 'patternsearch'
89
90          opt = optimset('Display',       'iter',...
91                         'PlotFcns',      {@psplotbestf @psplotbestx},...
92                         'OutputFcn',     {@fmincon_outfun},...
93                         'CompletePoll',  'on');
94          [p_estimate,fval,exitflag,output] = patternsearch(@(p)odefit(ut,u,p,X),etatmp
                ,[],[],[],[],[],[],[],opt);
95
96      case 'levenbergmarquardt'
97
98          opt = optimoptions('lsqnonlin');
99          opt = optimoptions(opt,...
100                            'Algorithm', 'levenberg-marquardt' , ...
101                            'Display','iter', ...
102                            'TolX', 1e-10, ...
103                            'TolFun', 1e-6, ...
104                            'FinDiffType','central', ...
105                            'TypicalX',-ones(size(etatmp,1),1),...
106                            'PlotFcns', {@optimplotx @optimplotfval @optimplotresnorm
                                  },...
107                            'OutputFcn',  @fmincon_outfun);
108
109         [p_estimate,fval,residual,exitflag,output,lambda,jacobian] = lsqnonlin(@(p)
                odefit(ut,u,p,X),etatmp,[],[],opt);
110
111
```

```
    case 'genetic'

        ObjectiveFunction=@(p)odefit(ut,u,p,X);
        nvars = n_eta;
        LB = -15*ones(1,nvars);
        UB = 15*ones(1,nvars);

        options = gaoptimset;
        options = gaoptimset(options,...
                             'Display',              'iter', ...
                             'Generations',          50, ...
                             'PopulationSize',       10,...
                             'TolFun',               1e-8,...
                             'UseParallel',          'never',...
                             'CrossoverFraction',    0.8,...
                             'PopInitRange',         [-1;1]);
        [eta,fval] = ga(ObjectiveFunction,nvars,[],[],[],[],LB,UB,[],options);


end

if biaszero == true
    p_estimate=[p_estimate; zeros(n_x,1)];
end

% Ergebnis des Trainings berechnen und anzeigen:
options = odeset('RelTol',1e-4,'AbsTol',1e-4,'MaxStep',7.5);

if strcmp(CTRNNForm,'classical')
    [T, Xtrain] = ode15s(@(t,x) ode_rhs_class(t,x,ut,u,p_estimate,n_x,n_u,n_H,
       calcSens),Tspan,IC,options); % Solve ODE
elseif strcmp(CTRNNForm,'alseyab')

    [T, Xtrain] = ode15s(@(t,x) ode_rhs_seyab(t,x,ut,u,p_estimate,n_x,n_u,n_H,
       calcSens),Tspan,IC,options); % Solve ODE
end
Y = C * Xtrain(:,1:n_x)';

figure
plot(T, X); hold on;plot(T,Xtrain(:,1:1),'--r');hold off
title('Plot of y as a function of time');
xlabel('Time'); ylabel('Y(t)');

    % Gütefunktion. Vorhersagefehler einer NN-Parameterschätzung berechnen
    function e = odefit(ut,u,p,X)

        if biaszero == true
            if strcmp(method,'genetic')
                p(nvars-2:end)=0;
                p=p';
            else
                p = [p; zeros(n_x,1)];
            end
        end

        options = odeset('RelTol',1e-4,'AbsTol',1e-4,'MaxStep',7.5);
        if strcmp(CTRNNForm,'classical')
            [T, Xtrain] = ode15s(@(t,x) ode_rhs_class(t,x,ut,u,p,n_x,n_u,n_H,calcSens
               ),Tspan,IC,options);
        elseif strcmp(CTRNNForm,'alseyab')
```

```
169             [T, Xtrain] = ode15s(@(t,x) ode_rhs_seyab(t,x,ut,u,p,n_x,n_u,n_H,calcSens
                    ),Tspan,IC,options);
170         end
171         Y = C * Xtrain(:,1:n_x)';
172 
173         try
174 
175 
176             %für lsqnonlin braucht man abweichende Form
177             if strcmp(method,'levenbergmarquardt')
178                 e = ((Y'-X));
179             else
180                 e = sum((Y'-X).^2);
181             end
182 
183 
184         catch
185              disp('NaN zurückgegeben')
186              e=NaN;
187         end
188     end
189 
190     % Nach jeder Iteration Zwischenwerte speichern
191     function stop  = fmincon_outfun(x, optimValues, state,varargin)
192         stop=false;
193         if strcmp(state,'iter')==1
194             if strcmp(method,'levenbergmarquardt')
195                 iterhistory.fval = [iterhistory.fval optimValues.resnorm];
196             else
197                 iterhistory.fval = [iterhistory.fval optimValues.fval];
198             end
199             iterhistory.param= [iterhistory.param x];
200             iterhistory.funccount= [iterhistory.funccount optimValues.funccount];
201         end
202     end
203 
204 end
```

Programmcode A.5: Matlab-Code zur Auswertung verschiedener trainierter neuronaler Netze.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Auswertung.m                                                          %
% Trainierte Netze verschiedener Komplexitäten in ihrer Qualität        %
% vergleichen                                                           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all

% Auszuwertende NN-Objekte
filenames = cell(2);
filenames{1,1}= 'h3d2';          filenames{1,2}= 'h3, d2.     R^2 = ';
filenames{2,1}= 'h3d3';          filenames{2,2}= 'h3, d3.     R^2 = ';
filenames{3,1}= 'h5d2';          filenames{3,2}= 'h5, d2.     R^2 = ';
filenames{4,1}= 'h5d5';          filenames{4,2}= 'h5, d5.     R^2 = ';
filenames{5,1}= 'h10d2';         filenames{5,2}= 'h10, d2.    R^2 = ';
filenames{6,1}= 'h10d5';         filenames{6,2}= 'h10, d5.    R^2 = ';
filenames{7,1}= 'h15d2';         filenames{7,2}= 'h15, d2.    R^2 = ';
filenames{7,1}= 'h30d10';        filenames{7,2}= 'h30, d10.   R^2 = ';
filenames{8,1}= 'h50d2';         filenames{8,2}= 'h50, d2.    R^2 = ';

tmp=csvread('dymola.csv');
sim_u = tmp(:,3);
sim_y = 1e-6 *tmp(:,2);
inputSeries = tonndata(sim_u,false,false);
targetSeries = tonndata(sim_y,false,false);


% Vergleich Trainingssatz (nicht Validierungsdaten) und Closeloop-Vorhersage des NN
figure('name','Trainingsergebnis: Vergleich NN-Vorhersage und Soll')
for i=1:length(filenames)
    S=load(filenames{i});
    loadednet=eval(['S.' filenames{i}]);
    [Xs,Xi,Ai,Ts] = preparets(loadednet,inputSeries,{},targetSeries);
    subplot(ceil(size(filenames,1)/2),2,i);

    nn_out=loadednet(Xs,Xi,Ai);
    plot(cell2mat(nn_out));
    hold on
    plot(sim_y,'r');
    hold off
    xlim([0 length(nn_out)])
    corr=corrcoef(cell2mat(nn_out),sim_y( length(sim_y)-length(Xs)+1:length(sim_y)));
    maxErr=max(abs(cell2mat(gsubtract(Ts,nn_out))));
    title([filenames{i,2} num2str(corr(2,1)) '    maxErr = ' num2str(maxErr)]);
end


% Vorhersagefehler des Closeloop-NN in Bezug auf die Validierungsdatenauswerten
figure('name','Trainingsergebnis: Fehler zwischen NN-Vorhersage und Soll')
for i=1:length(filenames)
    S=load(filenames{i});
    loadednet=eval(['S.' filenames{i}]);
    [Xs,Xi,Ai,Ts] = preparets(loadednet,inputSeries,{},targetSeries);
    subplot(ceil(size(filenames,1)/2),2,i);

    nn_out=loadednet(Xs,Xi,Ai);
    plot(cell2mat(gsubtract(Ts,nn_out)));
    xlim([0 length(nn_out)])
    corr=corrcoef(cell2mat(nn_out),sim_y( length(sim_y)-length(Xs)+1:length(sim_y)));
    maxErr=max(abs(cell2mat(gsubtract(Ts,nn_out))));
```

```
    title([filenames{i,2} num2str(corr(2,1)) '    maxErr = ' num2str(maxErr)]);
end


% Auswertung mit Validierungssatz, nicht Trainingssatz
tmp=csvread('dymola_val.csv');
sim_time = tmp(:,1);
sim_u = tmp(:,3);
sim_y = 1e-6 *tmp(:,2);
inputSeries = tonndata(sim_u,false,false);
targetSeries = tonndata(sim_y,false,false);


% Datenbasis
figure('name','Datenbasis')
subplot(2,1,2); plot(sim_y); title('Systemantwort für Eingangsgröße'); subplot(2,1,1)
    ; plot(cell2mat(inputSeries),'r');title('Eingangsgröße für Validierungsfall')


% NN-Vorhersage (closeloop) und Validierungsdaten vergleichen
figure('name','Validierung: Vergleich zwischen NN-Vorhersage und Validierungssatz')
for i=1:length(filenames)
    S=load(filenames{i});
    loadednet=eval(['S.' filenames{i}]);
    [Xs,Xi,Ai,Ts] = preparets(loadednet,inputSeries,{},targetSeries);
    subplot(ceil(size(filenames,1)/2),2,i);

    nn_out=loadednet(Xs,Xi,Ai);
    plot(cell2mat(nn_out));
    RESULTS(i,:)=cell2mat(nn_out)';
    hold on
    %plot(sim_y,'r');
    plot(sim_y( length(sim_y)-length(Xs)+1:length(sim_y)),'r')
    hold off
    xlim([0 length(nn_out)])
    corr=corrcoef(cell2mat(nn_out),sim_y( length(sim_y)-length(Xs)+1:length(sim_y)));
    maxErr=max(abs(cell2mat(gsubtract(Ts,nn_out))));
    title([filenames{i,2} num2str(corr(2,1)) '    maxErr = ' num2str(maxErr)]);
end


% Vorhersagefehler in Bezug auf Validierungsdaten auswerten
figure('name','Validierung: Fehler zwischen NN-Vorhersage und Validierungssatz')
for i=1:length(filenames)
    S=load(filenames{i});
    loadednet=eval(['S.' filenames{i}]);
    [Xs,Xi,Ai,Ts] = preparets(loadednet,inputSeries,{},targetSeries);
    subplot(ceil(size(filenames,1)/2),2,i);

    nn_out=loadednet(Xs,Xi,Ai);
    plot(cell2mat(gsubtract(Ts,nn_out)));
    xlim([0 length(nn_out)])
    corr=corrcoef(cell2mat(nn_out),sim_y( length(sim_y)-length(Xs)+1:length(sim_y)));
    maxErr=max(abs(cell2mat(gsubtract(Ts,nn_out))));
    title([filenames{i,2} num2str(corr(2,1)) '    maxErr = ' num2str(maxErr)]);
end
```

Programmcode A.6: Matlab-Code zur Umwandung eines NARX-Netzes in ein stetiges Ersatzmodell.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% deriveEq.m                                                            %
% Ableiten eines stetigen Ersatzmodells aus einem NARX-Netzwerk  %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
load h5d2;
net2 = h5d2;

nHidden=5; % Anzahl der Neuronen in der versteckten Schicht

% Zeitschrittweite der Daten, mit denen das Netz trainiert wurde
dt=1;

% Gewichte aus dem Netz extrahieren
W_hiddenZuOut=net2.LW{2,1};
W_OutZuYIn=net2.LW{1,2};
W_UInZuHidden=net2.IW{1};
W_b_hidden=net2.b{1};
W_b_out=net2.b{2};

w_IH1 = W_UInZuHidden(:,1);
w_IH2 = W_UInZuHidden(:,2);
w_OH1 = W_OutZuYIn(:,1);
w_OH2 = W_OutZuYIn(:,2);
b_H = net2.b{1};
b_O = net2.b{2};

% Gewichte des Netzes zu einer stetigen Gleichung weiterverarbeiten
H=cell(nHidden,1);
for i=1:nHidden
H{i} = ['tanh( u*(' num2str(w_IH1(i) + w_IH2(i)) ') + u_p*(' num2str(-dt * w_IH1(i) -
     2*dt * w_IH2(i)) ') + der(u_p)*(' num2str(dt^2 * w_IH2(i)) ') + y*(' num2str(
    w_OH1(i) + w_OH2(i)) ') + y_p*('  num2str(-dt * w_OH1(i) - 2*dt * w_OH2(i)) ') +
    der(y_p)*('  num2str(dt^2 * w_OH2(i)) ') + (' num2str(b_H(i)) ') )'];
end

resstr= ['y = ' num2str(b_O)];
for i=1:nHidden
resstr = [resstr ' + (' num2str(W_hiddenZuOut(i)) ') * ' H{i}]  ;
end
disp(resstr)
```

Programmcode A.7: Modelica/Optimica-Code zur Zeitoptimierung einer Steuerung.

```
optimization DI_opt(objective=finalTime, objectiveIntegrand = 0, startTime=0,
    finalTime(free=true,min=1,initialGuess=134))

  Real x1(start=0,fixed=true);
  Real x2(start=0,fixed=true);
  Real u1_p(start=0,fixed=true);
  Real u1(start=0,fixed=true);
  input Real u1_pp(free=true, min= -0.0025, max=0.0025);

equation

  // Ersatzmodell
  der(x1) = 0.35151 * tanh( -0.78595*x1-1.4085*x2 + -0.89663*u1_p + -1.2267) + 0
      .32827 * tanh( -0.36125*x1-0.76245*x2 + 1.1848*u1_p + -0.25058) +  + 0.38134;

  der(x2) = -0.14652 * tanh( -0.78595*x1-1.4085*x2 + -0.89663*u1_p + -1.2267) + 1
      .5237 * tanh( -0.36125*x1-0.76245*x2 + 1.1848*u1_p + -0.25058) +  + 0.26419;

  der(u1)=u1_p;
  der(u1_p)=u1_pp;

constraint

  u1(0)=0;
  u1_p(0)=0;

  u1_pp(finalTime)=0;
  u1(finalTime)=1;
  u1_p(finalTime)=0;

  u1_p <= 0.03;
  x1 >= -0.15;
  u1 >= 0;


end DI_opt;
```

Programmcode A.8: Matlab-Code zur Optimierung der Steuerung mit einem Kollokationsverfahren in JModelica.org.

```
import numpy as N
import matplotlib.pyplot as plt
from pymodelica import compile_fmu
from pyfmi import load_fmu
from pyjmi import transfer_optimization_problem
from pyjmi.optimization.casadi_collocation import *

vdp = transfer_optimization_problem("DI_opt","C:/vdp122.mop")


# Anfangswertschätzung aus grober Lösung gewinnen
opts = vdp.optimize_options()
opts['n_e']=5
opts['n_cp'] = 2
opts['discr'] = 'LGR'
opts['result_mode'] = 'element_interpolation'
opts['n_eval_points'] = 20
opts['hs'] = ([ 0.09,  0.076,  0.087,  0.731,  0.016])
opts['blocking_factors'] = [1,1,1,1,1]
res = vdp.optimize(options=opts)

# Eigentliche Lösung des Problems
opts = vdp.optimize_options()
opts['n_e'] = 6
opts['n_cp'] = 9
opts['discr'] = 'LGR'
opts['result_mode'] = 'element_interpolation'
opts['n_eval_points'] = 10

opts['IPOPT_options']['max_iter'] = 50000
opts['IPOPT_options']['tol'] = 1e-15
opts['init_traj']=res.result_data

res = vdp.optimize(options=opts)

# Ergebnisse anzeigen
y=res['y']
y_p=res['y_p']
u_p=res['u_p']
u_pp=res['u_pp']
y_pp=res['y_pp']
u=res['u']
t=res['time']
y_unscaled=res['y_unscaled']
u_unscaled=res['u_unscaled']

plt.figure(1)
plt.clf()

plt.subplot(811)
plt.plot(t,y)
plt.grid()
plt.ylabel('y')

plt.subplot(812)
plt.plot(t,y_p)
plt.grid()
```

```
58 plt.ylabel('y_p')
59
60 plt.subplot(813)
61 plt.plot(t,y_pp)
62 plt.grid()
63 plt.ylabel('y_pp')
64
65
66 plt.subplot(814)
67 plt.plot(t,u)
68 plt.grid()
69 plt.ylabel('u')
70
71
72 plt.subplot(815)
73 plt.plot(t,u_p)
74 plt.grid()
75 plt.ylabel('u_p')
76
77 plt.subplot(816)
78 plt.plot(t,u_pp)
79 plt.grid()
80 plt.ylabel('u_pp')
81
82 plt.subplot(817)
83 plt.plot(t,y_unscaled)
84 plt.grid()
85 plt.ylabel('y_unscaled')
86
87 plt.subplot(818)
88 plt.plot(t,u_unscaled)
89 plt.grid()
90 plt.ylabel('u_unscaled')
91
92 plt.show()
93
94 # Ergebnisse exportieren
95 import exportCsv
96 from exportCsv import write_csv
97 write_csv(res.result_data,'result_export.csv')
```

Programmcode A.9: Matlab-Code zur Optimierung der Steuerung mit direktem Schießen.

```
function [iterhistory] = optimize2
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% optimize2                                                              %
% Zeitdiskrete Optimierung der Steuerung auf Basis eines NARX-           %
% Netzwerks                                                              %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Tfinal = 110;
h=2;
T= 0:h:Tfinal-h;
u0 = 0.5*ones(Tfinal / h,1);

LB=0;           % untere Grenze
UB=1;           % obere Grenze
IC = [0 0];     % Anfangsbedingungen
desired_h=1;    % Samplingzeit des NN

iterhistory.u = [];
iterhistory.fval = [];
iterhistory.cviol = [];
uLast = [];

N = Tfinal / h;
Umax = UB * ones(N,1);
Umin = LB * ones(N,1);


options = optimset( 'LargeScale',       'on', ...
                    'Display',          'iter', ...
                    'TolX',             0.0001, ...
                    'TolFun',           0.0001, ...
                    'TolCon',           0.0001, ...
                    'MaxFunEvals',      200000, ...
                    'MaxIter',          200000, ...
                    'PlotFcns',         { @optimplotx @optimplotfval
                       @optimplotconstrviolation}, ...
                    'Algorithm',        'sqp',...
                    'FinDiffType',      'central',...
                    'OutputFcn',        @fmincon_outfun, ...
                    'UseParallel',      'always' ...
                                      );

% Optimierung starten
[u,fval,exitflag,output,lambda,grad,hessian] = fmincon(@obj_fun2,u0,[],[],[],[],Umin,
    Umax,@con_fun2, options, h, Tfinal,IC);

% Optimierungsergebnisse anzeigen
[tout, yout] = evaluate_system(T,u,IC,desired_h,Tfinal,h);
figure
subplot(3,1,1)
plot(tout,yout)
subplot(3,1,2)
Tneu= 0:desired_h:Tfinal-h;
u_NN = interp1(T,u,Tneu)';
plot(Tneu,u_NN); hold on; plot(T,u,'r')

subplot(3,1,3)
plot(Tneu,u_NN)

```

```
58
59      % Nach jeder Iteration wichtige Werte speichern
60      function stop  = fmincon_outfun(x, optimValues, state,varargin)
61          stop=false;
62          if strcmp(state,'iter')==1
63              iterhistory.fval = [iterhistory.fval optimValues.fval];
64              iterhistory.u    = [iterhistory.u x(:)];
65              iterhistory.cviol= [iterhistory.cviol optimValues.constrviolation];
66          end
67      end
68
69      % Implementierung der Kostenfunktion
70      function f = obj_fun2(u,h,Tfinal,IC)
71
72          % Wurde dieses u schon ausgewertet?
73          if ~isequal(u,uLast)
74              % Wenn nicht, dann speise aktuelles u in das Ersatzmodell ein
75              % und erhalte Ergebnis
76              [tout,yout] = ComputeSystemOutput(u,h,Tfinal,IC);
77              uLast = u;
78          end
79
80          % Berechne den Wert der Kostenfunktion:
81          f = 100*(u-1)' * (u-1) ;
82
83      end
84
85      % Implementierung der Beschränkungsfunktion
86      function [ g, psi ] = con_fun2( u, h, Tfinal,IC )
87
88          % Wurde dieses u schon ausgewertet?
89          if ~isequal(u,uLast)
90              % Wenn nicht, dann speise aktuelles u in das Ersatzmodell ein
91              % und erhalte Ergebnis
92              [tout,yout] = ComputeSystemOutput(u,h,Tfinal,IC);
93              uLast = u;
94          end
95
96          % Rechne gröbere Zeitsegmentierung der Optimierung in feinere um
97          Tneu= 0:desired_h:Tfinal-h;
98          u_NN = interp1(T,u,Tneu)';
99
100         % Wert der Ungleichungsnebenbedingungen berechnen
101         % 1. sigma <= -150 MPa, 2. du/dt <= 10%/s
102         g = [-yout(:,1) - 150; abs((diff([0;u_NN]))) - 0.1];
103
104         %Gleichungsnebenbedingung nicht notwendig:
105         psi = [];
106
107     end
108
109     % Systemantwort für bestimmtes u aus dem NARX-Netz ableiten
110     function  [tout, yout] = ComputeSystemOutput(u,h,Tfinal,IC)
111         T = [0:h:Tfinal-h];
112         [tout, yout]= evaluate_system(T,u,IC,desired_h,Tfinal,h);
113     end
114
115     function [tout, yout] = evaluate_system( T, u,IC,varargin )
116
117         % Soll die Optimierung anhand eines CTRNN oder eines NARX erfolgen?
```

```
118         % Standard ist NARX.
119         if nargin <= 3
120 
121         % löse DGL von 0 bis Tfinal mit h und TU
122         [tout, yout] = ode23(@(t,x) odesystem(t,x,T,u),T,IC);
123 
124         else
125             % Lade neuronales Netz
126             nnet = load('h5d2');
127             nnet=nnet.h5d2;
128 
129             % u umrechnen in gewünschte Sampling-Zeit:
130             h_desired=varargin{1};
131             Tfinal=varargin{2};
132             h = varargin{3};
133             Tneu= 0:h_desired:Tfinal-h;
134             u_NN = interp1(T,u,Tneu);
135 
136             % nötig für die richtigen Anfangsbedingungen im NN
137             u_NN = [0 0 u_NN];
138 
139             inputSeries = tonndata(u_NN',false,false);
140             targetSeries = tonndata(0 * u_NN',false,false);
141 
142             % NN simulieren
143             [Xs,Xi,Ai,Ts] = preparets(nnet,inputSeries,{},targetSeries);
144             nn_out=nnet(Xs,Xi,Ai);
145             nn_out = cell2mat(nn_out);
146 
147             tout = Tneu';
148             yout = nn_out';
149 
150         end
151     end
152 
153 end
```

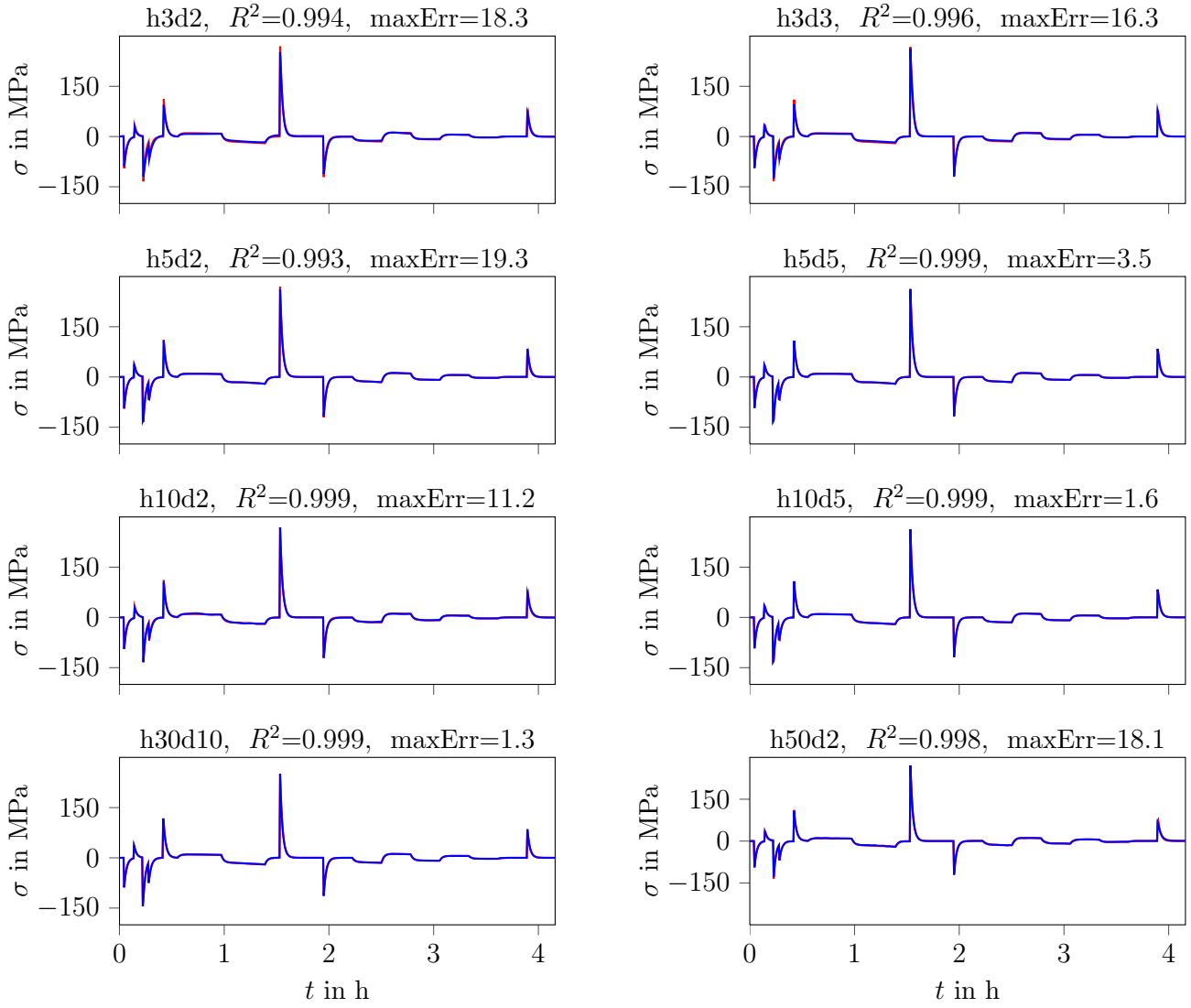


Abbildung A.1.: Trainingsergebnisse verschiedener NARX-Netzwerke für die Mischungsstrecke.

i	α_i	$\beta_{i,1}$	$\beta_{i,2}$	$\beta_{i,3}$	$\delta_{i,1}$	$\delta_{i,2}$	$\delta_{i,3}$	κ_i
0	−0.17482	–	–	–	–	–	–	–
1	−0.20619	−0.40786	1.8229	−1.415	0.010988	0.40954	−0.42053	1.3872
2	−0.33921	−0.27915	−0.01618	0.29534	−0.21399	0.33976	−0.12576	−0.35232
3	1.0388	−0.056448	−0.21432	0.27077	0.59799	−0.5068	−0.09119	−0.71802
4	1.8471	−0.05514	−0.22738	0.28252	0.44054	−0.35417	−0.086372	0.66597
5	−0.19763	−1.5553	3.6335	−2.0781	−3.4341	5.7108	−2.2767	3.3681

Tabelle A.1.: Koeffizienten α, β, δ und κ zur Ableitung eines stetigen Ersatzmodells der Mischungsstrecke.

Stichwortverzeichnis