Technische Universität Dresden

Fakultät Elektrotechnik und Informationstechnik

Institut für Regelungs- und Steuerungstheorie

Diplomarbeit

Vergleich von Modellbildungsansätzen im Vertical-Gradient-Freeze-Kristallzüchtungsprozess

vorgelegt von: Christoph Blümbott

geboren am: 8. Februar 1996 in Frankenberg (Sachsen)

zum Erlangen des akademischen Grades

Diplomingenieur

(Dipl.-Ing.)

Betreuer:	Dipl.-Ing. Stefan Ecklebe
Verantwortlicher Hochschullehrer:	Prof. Dr.-Ing. habil. Dipl.-Math. K. Röbenack
Tag der Einreichung:	16. März 2020

Bitte ersetzen Sie diese Seite vor dem Binden mit der Aufgabenstellung.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die von mir am heutigen Tage dem Prüfungsausschuss der Fakultät Elektrotechnik und Informationstechnik eingereichte Diplomarbeit zum Thema

Vergleich von Modellbildungsansätzen im Vertical-Gradient-Freeze-Kristallzüchtungsprozess

selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Schriften entnommen sind, wurden als solche kenntlich gemacht.

Dresden, 16. März 2020 Christoph Blümbott

Kurzfassung

Der Vertical-Gradient-Freeze (VGF)-Kristallzüchtungsprozess ist ein Verfahren zur Herstellung von Einkristallen, die in der Halbleiterindustrie benötigt werden. Der Einkristall wird durch ein sich veränderndes Temperaturfeld gezüchtet. Die dafür benötigten Heizer sind um den Tiegel herum angebracht und müssen geregelt werden. Es handelt sich dabei um eine modellprädiktive Regelung. Bekannte Informationen sind nur die Temperaturen an den Heizern und die Heizleistungen bzw. -temperaturen. Mittels der zwei Systemidentifikationsansätze sparse identification of nonlinear dynamics with control (SINDYc) und ordinary differential equation network (ODEnet) soll das System im Tiegel identifiziert werden. SINDYc basiert auf der Minimierung über ein Kostenfunktional aus Ansatzfunktionen und das ODEnet auf der Verbindung eines künstlichen neuronalen Netzes mit einem Differentialgleichungslöser. Es erfolgt die Implementierung der beiden Ansätze und der Test mit Testsystemen. Bei SINDYc fallen diese Tests sehr gut aus, wohingegen die Ergebnisse beim ODEnet nicht immer zufriedenstellend sind. Beim Test mit realen Prozessdaten, können beide Ansätze die Systeme nicht optimal identifizieren aber eine ungefähre Systemidentifikation ist möglich. Insbesondere das ODEnet erzielt bessere Ergebnisse als SINDYc. Die Gründe dafür liegen in der Auswahl der Trainingsdaten und dem in SINDYc genutzten least absolute shrinkage and selection operator (LASSO)-Regression.

Abstract

The VGF crystal growth process is a process for the production of single crystals that are required in the semiconductor industry. The single crystal is grown by a changing temperature field. The heaters required for this are attached to the crucible and must be controlled. The controller is a model predictive controller. Known information is only the temperatures at the heaters and the heating outputs or temperatures. The system is to be identified in the crucible using the two system identification approaches SINDYc and ODEnet. SINDYc is based on the minimization via a cost functional from approach functions and the ODEnet on the connection of an artificial neural network with a differential equation solver. The two approaches are implemented and tested using test systems. These tests are very good at SINDYc, whereas the results from ODEnet are not always satisfactory.. When testing with real process data, both approaches cannot optimally identify the systems, but an approximate system identification is possible. ODEnet in particular achieves better results than SINDYc. The reasons for this lie in the selection of the training data and the LASSO algorithm used in SINDYc.

Inhaltsverzeichnis

Abkürzungsverzeichnis

DGL	Differentialgleichung
KNN	künstliches neuronales Netz
LASSO	least absolute shrinkage and selection operator
LSTM	langes Kurzzeitgedächtnis
MPC	modellprädiktive Regelung
MQA	mittlere quadratische Abweichung
ODEnet	ordinary differential equation network
QA	quadratische Abweichung
ReLU	rectified linear unit
RNN	rekurenntes neuronales Netz
SINDY	sparse identification of nonlinear dynamics
SINDYc	sparse identification of nonlinear dynamics with control
VGF	Vertical-Gradient-Freeze

Abbildungsverzeichnis

Tabellenverzeichnis

Symbolverzeichnis

Allgemein

e	Regeldifferenz $e = y - \hat{y}$
η	Anzahl der Eingänge
f	Systemdynamik
m	Anzahl der Datenpunkte
μ	Mittelwert
n	Anzahl der Zustände
n_C	Steuerhorizont
n_P	Prädiktionshorizont
s	Standardabweichung
r	Abstand zwischen den benutzten Datenpunkten
t	Zeit
t_s	skalierte Zeit
Δt	Zeitdifferenz
$\boldsymbol{u} \in \mathbb{R}^\eta$	Eingangsvektor
$u(k)$	Steuergröße am Abtastzeitpunkt k
w	Führungsgröße
$\boldsymbol{x}(t) \in \mathbb{R}^n$	Zustandsvektor
$\Delta\boldsymbol{x}(t) \in \mathbb{R}^n$	Differenz des Zustandsvektors vom Referenzzustand
$x_s(t)$	skalierte Daten
y	Regelgröße
$\hat{y}$	Vorhersage des Prädiktors

SINDYc

α	Grad des Polynoms
β_p	wahre Koeffizienten bei der LASSO-Regression
$\boldsymbol{\beta} \in \mathbb{R}^q$	Vektor der wahren Koeffizienten bei der LASSO-Regression
$\hat{\beta}$	geschätzte Koeffizienten bei der LASSO-Regression
$\hat{\boldsymbol{\beta}} \in \mathbb{R}^q$	Vektor der geschätzten Koeffizienten bei der LASSO-Regression
ϵ_i	Residuen bei der LASSO-Regression zum Zeitpunkt t_i
$\boldsymbol{\epsilon} \in \mathbb{R}^m$	Vektor der Residuen bei der LASSO-Regression
D	Potenzfaktor

ΔD	Potenzunterschied
λ	1-Norm-Penalisierungsparameter
k	Anzahl der Variablen der Polynome
p_k^α	Polynom des Grades α mit k Variablen
$\boldsymbol{p}_k^\alpha \in \mathbb{R}^{1\times\frac{(k+\alpha)!}{2k!\alpha!}}$	Vektor der Polynomkomponenten des Grades α mit k Variablen
$\boldsymbol{P}_k^\alpha \in \mathbb{R}^{m\times\frac{(k+\alpha)!}{2k!\alpha!}}$	Matrix von Polynomauswertungen an den Zeitpunkten t
$\boldsymbol{\Theta} \in \mathbb{R}^{m\times l}$	Matrix mit Ansatzfunktionen
v_{iq}	Modellvariablen bei der LASSO-Regression zum Zeitpunkt t_i
$\boldsymbol{V} \in \mathbb{R}^{m\times q}$	Matrix der Modellvariablen bei der LASSO-Regression
$\boldsymbol{X} \in \mathbb{R}^{m\times n}$	Matrix der Zustandsdaten
$\dot{\boldsymbol{X}} \in \mathbb{R}^{m\times n}$	Matrix der Ableitungen der Zustandsdaten
$\boldsymbol{\Xi} \in \mathbb{R}^{l\times n}$	Matrix mit Koeffizienten
$\boldsymbol{\xi}_k \in \mathbb{R}^l$	Vektor mit Koeffizienten
$\boldsymbol{\Upsilon} \in \mathbb{R}^{m\times\eta}$	Matrix der Eingangsdaten
z_i	Zielgröße bei der LASSO-Regression zum Zeitpunkt t_i
$\boldsymbol{z} \in \mathbb{R}^m$	Zielgrößenvektor bei der LASSO-Regression

ODEnet

$\boldsymbol{a}$	adjunkte
$\boldsymbol{h}$	verborgener Zustand oder auch verborgene Schicht des KNN
h	Anzahl verborgener Schichten im ODEnet bzw. Schrittweite
L	Kostenfunktional
o_j	Aktivierung
φ	Aktivierungsfunktion
θ	Netzparameter
υ	Netzeingabe
w_{ij}	Gewichte von Neuron i zu Neuron j
x_0	Bias
χ_i	Eingänge in ein Neuron

Kapitel 1

Einleitung

1.1 Motivation

Die übergeordnete Motivation für diese Arbeit ist die Optimierung des Vertical-Gradient-Freeze (VGF)-Kristallzüchtungsprozess. Der Prozess ist hauptsächlich darauf ausgelegt, einem Medium durch Heizen und Erstarren eine bestimmte Kristallstruktur zu geben. Das Problem des Heizens und Erstarrens wird in dieser Arbeit behandelt.

Beim VGF-Kristallzüchtungsprozess wird die Bildung der Kristallstruktur durch ein sich bewegendes Temperaturfeld sichergestellt. In einem Tiegel befindet sich das geschmolzene Ausgangsmaterial, in das ein Keimkristall eingebracht wird. Durch die Veränderung des Temperaturfeldes im Tiegel, erstarrt das Ausgangsmaterial wieder am Keimkristall. Das Temperaturfeld wird dabei so gesteuert, dass es im Tiegel nach oben wandert und die gesamte Schmelze erstarrt. Da es hierbei um einen sehr heißen Prozess handelt, können zur Temperaturmessung keine Sensoren in den Tiegel eingebracht werden. Des Weiteren lassen auch die Reinheitsanforderungen und die Reaktivität der Materialien keine Sensoren zu. Demzufolge stehen für die Temperaturmessung lediglich die Informationen der Heizelemente am Mantel und an den Stirnseiten des Tiegels zur Verfügung. Die Aufgabenstellung ist, ein Anlagenmodell in Zustandsraumdarstellung zu erstellen, das in der modellprädiktive Regelung (MPC) verwendet werden kann. Dabei soll ein Vergleich verschiedener ausgewählter Modellierungsansätze im Hinblick auf Datenmengen, Erstellungsaufwand und Zuverlässigkeit auf Basis von realen Messdaten durchgeführt werden [31], [30].

Durch das Heizen und Erstarren bildet sich an einem bestimmten Punkt eine Phasengrenze aus, an der die Kristallstruktur gebildet wird. Zum optimalen Heizen und Erstarren muss der Temperaturverlauf im Medium bekannt sein. Eine Messung der Temperatur im Tiegel ist jedoch aufgrund der hohen Temperatur nicht möglich. Demzufolge ist ein Regeln auf Grundlage genauer Messungen im Tiegel nicht möglich, da nur die Informationen der Heizer und von Elementen außerhalb des Tiegels zur Verfügung stehen. Um gleichwohl eine möglichst gute Regelung zu ermöglichen ist das Wissen um ein mathematisches System des Temperaturverlaufs im Tiegel erforderlich. Eine

Berechnungsmethode ist beispielsweise die Finite-Elemente-Methoden-Simulation, die aber durch lange Rechenzeiten eher langsam ist, denn für die Berechnung des Temperaturverlaufs im Medium muss ein großer Rechenaufwand betrieben werden. Diese Methode erfordert genaues Wissen der physikalischen und chemischen Eigenschaften im Tiegel und langer Rechenzeiten.

Um das mathematische Modell des Temperaturverlaufs zu finden werden verschiedene, in der Literatur beschriebene, Ansätze getestet. Dabei wurden einige Ansätze schon im Gebiet der Regelungstechnik getestet, andere sind hingegen noch neu. Da es sich um ein nichtlineares System handelt, müssen die Ansätze auch solche Systeme zufriedenstellend identifizieren können.

1.2 Präzisierung der Aufgabenstellung

Aufgrund der Vielzahl von Ansätzen zur Systemidentifikation, muss eine Auswahl getroffen werden. Dafür wird in einem ersten Schritt eine ausführliche Literaturrecherche durchgeführt, die verschiedene Identifikationsansätze auf die Eignung und Implementierbarkeit prüft. Dabei sollen auch Vor- und Nachteile gegenübergestellt werden und darauf basierend die Entscheidung der Eignung für das konkrete Problem im VGF-Kristallzüchtungsprozess getroffen werden.

Nach der Wahl der Identifikationsansätze erfolgt in einem zweiten Schritt deren Implementierung. Hierfür wird die Programmiersprache *Python* verwendet, da diese durch leichte Programmierbarkeit, Verständlichkeit und komfortable Erweiterung mit Bibliotheken besonders geeignet ist und am Institut für Regelungs- und Steuerungstheorie etabliert ist. Um die programmierten Ansätze während und nach der Implementierung ausgiebig zu testen, sind lineare und nichtlineare Beispielsysteme auszuwählen und zu implementieren. Hierdurch soll festgestellt werden, ob sich die Ansätze generell zur Anwendung in der Regelungstechnik eignen, wie gut die Ergebnisse bei bekannten Systemen sind und wie auf verschiedene Parametervariationen reagiert wird. Diese Erkenntnisse können dann zur Optimierung der Identifikation des VGF-Kristallzüchtungsprozess genutzt werden, ohne das die Messdaten der realen Anlage genutzt werden müssen.

Im dritten Schritt soll die Systemidentifikation der VGF-Kristallzüchtungsprozesses erfolgen. Dafür werden die Messdaten vorverarbeitet und anschließend in die Identifikationsansätze eingegeben. Ein optimales Ergebnis wäre ein Zustandsraumsystem, dass den Temperaturverlauf im Tiegel gut beschreibt.

Abschließend werden die verschiedenen Ergebnisse auf Abhängigkeiten vom gewählten Identifikationsansatz und dessen Parametern getestet. Hierbei soll auch ein Vergleich der Ansätze stattfinden und die Eignung für den VGF-Kristallzüchtungsprozess evaluiert werden. Sollte dies der Fall sein, könnten die gefundenen Modelle als Grundlage für einen modellprädiktiven Regler verwendet werden.

1.3 Gliederung der Arbeit

Zu Beginn des Kapitels 2 erfolgt eine Einführung in den VGF-Kristallzüchtungsprozess und die MPC. Dabei stehen die Anforderungen und Besonderheiten des VGF-Kristallzüchtungsprozess in Bezug auf den Temperaturverlauf und dessen Regelung im Vordergrund.

In den folgenden Abschnitten wird der Begriff der Systemidentifikation und mögliche Ansätze, die sich aus der Literaturrecherche herauskristallisiert haben, kurz erklärt. Daran anschließend wird mit sparse identification of nonlinear dynamics (SINDY) ein möglicher Ansatz ausführlich betrachtet. Dieser Ansatz wird für die regelungstechnische Anwendung um Eingänge erweitert und heißt sparse identification of nonlinear dynamics with control (SINDYc).

Ein zweiter Ansatz, das ordinary differential equation network (ODEnet), der als Grundlage ein künstliches neuronales Netz (KNN) nutzt, wird im nächsten Abschnitt eingeführt. Dabei wird auch auf die Besonderheiten der Berechnung der für diese Methode benötigten Funktionen eingegangen. Außerdem erfolgt an dieser Stelle eine kleine Einführung in KNN, deren Grundlagen für das Verstehen der Arbeit erforderlich sind.

In Kapitel 3 erfolgt die Implementierung der Algorithmen, sowie die Auswahl von Testsystemen und deren Tests mit den beiden Systemidentifikationsansätzen. Dafür werden ausgewählte Testsysteme vorgestellt. Daraufhin werden die Implementierung der beiden Systemidentifikationsansätze erklärt. Hier erfolgen wichtige Erläuterungen zu den genutzten Bibliotheken und zur Datenvorverarbeitung. Des Weiteren erfolgt die Darstellung der Ergebnisse der Systemidentifikationsansätze bei den Testsystemen. Ein Gütekriterium zur Bewertung der Ergebnisse wird eingeführt und die Darstellung der Einflüsse der verschiedenen Parameter beim SINDYc-Algorithmus wird evaluiert. Auch beim ODEnet-Ansatz werden ausgewählte Parameter variiert und dargestellt. Abschließend erfolgt ein Vergleich der beiden Ansätze.

In Kapitel 4 werden zwei gemessene Systeme aus realen Anlagendaten mit den beiden Systemidentifikationsansätzen identifiziert und Testmessungen prädiziert. Es erfolgt ein Vergleich der Referenzmessungen zu den Prädiktionen und die Kombination der beiden Ansätze.

Eine Zusammenfassung der Ergebnisse und mögliche weitere zu untersuchende Fragestellungen werden in Kapitel 5 dargestellt.

Kapitel 2

Vorbetrachtungen

2.1 Der Vertical-Gradient-Freeze-Kristallzüchtungsprozess

Einer unter mehreren Kristallzüchtungsverfahren ist der VGF-Kristallzüchtungsprozess. Neben diesem Verfahren ist noch das Czochralski-Verfahren zu nennen, das gemeinsam mit dem Zonenschmelzverfahren, das bekannteste unter den Züchtungsverfahren ist. Der VGF-Kristallzüchtungsprozess basiert auf dem Vertical-Bridgman-Verfahren. Im Unterschied zum Vertical-Bridgman-Verfahren findet keine mechanische Bewegung von Tiegel oder Ofen statt. Vielmehr wird der Prozess beim dem VGF-Kristallzüchtungsprozess durch ein sich zeitlich und örtlich veränderndes Temperaturfeld realisiert [14].

In einem Tiegel wird ein Keimkristall eingebracht und mittels Widerstandsheizern an Mantel und Stirnseite das Halbleitermaterial aufgeschmolzen. Dabei wird auch der Impfkristall teilweise geschmolzen. Durch das Bewegen des Temperaturfeldes wird die Schmelze so abgekühlt, dass ein Einkristall entsteht. Dieser Verlauf beginnt am Boden des Tiegels (am Impfkristall) und setzt sich langsam nach oben fort. Die Kristallisationsgeschwindigkeit wird durch die Geschwindigkeit der Verschiebung des Temperaturfeldes bestimmt [6]. Eine Darstellung des Querschnitts des Tiegels ist in Abbildung 1 gegeben.

Der VGF-Kristallzüchtungsprozess findet insbesondere Anwendung in der Herstellung von Halbleitern für die Optoelektronik (LED- und Lasertechnik), die Hochfrequenztechnik, die Solartechnik und die Telekommunikationstechnik. Hier werden Verbindungen Galliumarsenid (GaAs), Indiumphosphid (InP) und Galliumphosphid (GaP) verwendet. Der im VGF-Kristallzüchtungsprozess entstandene Kristall zeichnet sich durch eine geringe Defektdichte aus [14].

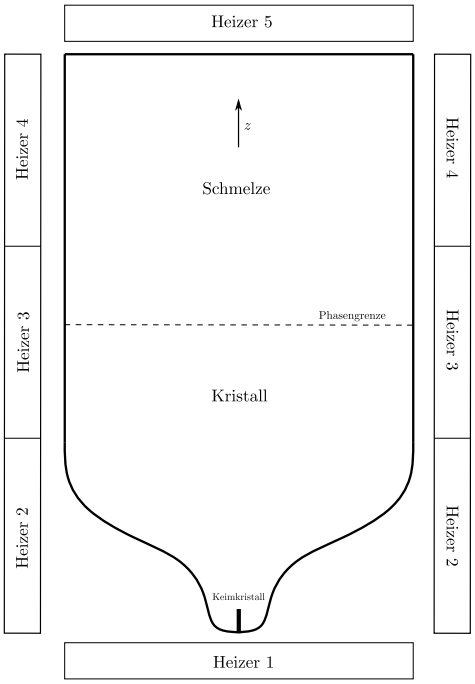

Abbildung 1 – Querschnitt durch den Tiegel, wie er beim VGF-Kristallzüchtungsprozess verwendet wird. An den Stirnseiten und am Mantel befinden sich die Heizer.

2.2 Modellbasierte prädiktive Regelung

Die MPC ist eine moderne Methode zum Regeln komplexer, zumeist multi-variabler Prozesse. Die Methode entstand in den 1970er-Jahren und wurde anfänglich in der chemischen und verfahrenstechnischen Industrie angewendet. Die MPC ist vergleichsweise rechenintensiv, weswegen sie sich insbesondere in Prozessen mit langsamen Abtastraten eignet, was in der chemischen und verfahrenstechnischen Industrie der Fall sein kann. Auch der VGF-Kristallzüchtungsprozess-Prozess fällt in dieses Gebiet. Durch die Entwicklung schneller Mikroprozessoren können heutzutage auch schnelle Prozesse mittels MPC geregelt werden [24], [8].

Der besondere Vorteil dieser Methode ist, dass Prozessbeschränkungen beim Reglerentwurf berücksichtigt werden können. Damit kann in den oben genannten Industrien eine höhere Ausbeute und bessere Ergebnisse als mit einem PID-Regler erzielt werden [24].

Der zu regelnde Prozess wird als dynamisches Modell benötigt, um das zukünftige Verhalten des Prozesses in Abhängigkeit von den Eingangssignales zu berechnen. Daraus wiederum können Eingangssignale für die Regelung berechnet werden, welche wiederum zu optimalen Ausgangssignalen führen. Es kann eine Online-Optimierung erfolgen, wodurch Prozess- und Stellgrößenbeschränkungen berücksichtigt werden können. Die Struktur eines Regelkreises mit einem modellprädiktiven Regler ist in Abbildung 2 dargestellt [24], [19].

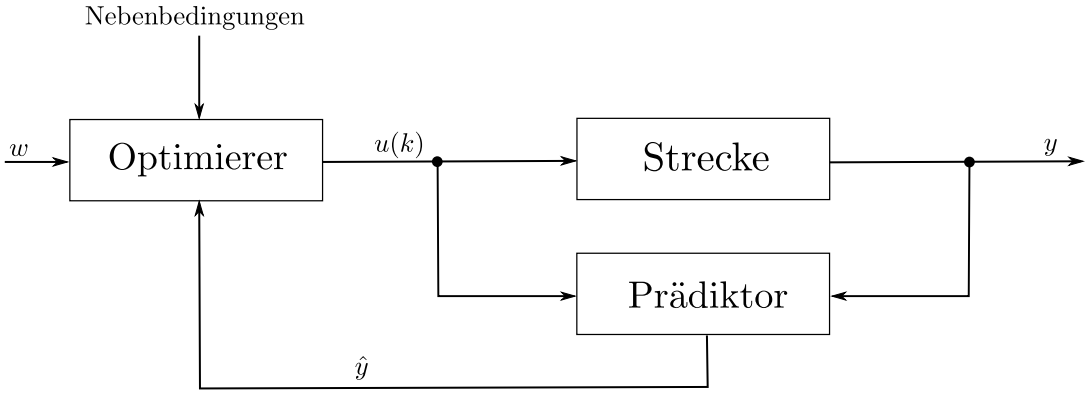

Abbildung 2 – Struktur eines prädiktiven Regelkreises, mit der Führungsgröße w, der Steuergröße u, dem Ausgang y und der Vorhersage $\hat{y}$.

Alle MPC-Regelungsalgorithmen haben die folgenden Hauptschritte gemeinsam, die im jedem Abtastintervall abgearbeitet werden:

Prädiktion: Unter der Annahme, dass sich die Steuergröße u in der Zukunft nicht verändert, wird eine Vorhersage des zukünftigen Verlaufs der Regelgröße y und der Regeldifferenz e durchgeführt. Auf Grundlage

- der aktuellen gemessenen und historischen Werte der Regelgröße y,
- der historischer Werte der Steuergröße u,
- der messbaren und historischen Größe z,
- eines gegebenen Sollverlaufs von w und
- eines gegebenen Modells für das dynamische Verhalten des Systems,

wird die Prädiktion berechnet. Der letzte Punkt der Liste ist derjenige, der in dieser Arbeit bereitgestellt werden soll: Ein mathematisches Modell, welches das dynamische Verhalten des Systems beschreibt.

Dynamische Optimierung: Während der Optimierung wird eine Folge zukünftiger Steuergrößenänderungen Δu über einen vorgegebenen Steuerhorizont n_c berechnet. Das Ziel hierbei ist die Minimierung der Regeldifferenz über den Prädiktionshorizont n_p. Das Ergebnis ist eine optimale Steuergößenfolge. Meistens ist diese nicht analytisch berechenbar, weswegen numerische Verfahren zur Berechnung genutzt werden. Um eine Minimierung durchzuführen ist ein Kostenfunktional nötig. Dieses beinhaltet in den meisten Fällen mindestens die Aspekte Regeldifferenz und die Stellaktivität. Während der dynamischen Optimierung können auch die Nebenbedingungen für die Steuer- und Regelgrößen beachtet werden, so zum Beispiel obere und untere Schranken.

Prinzip des gleitenden Horizonts: Während der dynamischen Optimierung werden eine ganze Folge zukünftiger Steuergrößen berechnet. Es wird jedoch nur das erste Element dieser Folge an den Prozess ausgegeben. Daraufhin wird der Horizont, d.h. die Datenvektoren u, y, e und w, um einen Zeitschritt nach vorne verschoben.

Korrektur der Vorhersage und Schließen des Regelkreises: In jedem Zeitschritt treffen neue Werte für die Regelgröße y ein. Damit wird die Vorhersage fortlaufend korrigiert und der Regelkreis geschlossen. Damit werden Störgrößen im Regelalgorithmus beachtet [8], [19].

Das Grundprinzip wird übersichtlich in Abbildung 3 gezeigt. Auch ein modellprädiktiver Regelkreis wird in Abbildung 2 dargestellt.

2.3 Systemidentifikation

Moderne Regelungsverfahren, wie in Zusammenhang mit dem VGF-Kristallzüchtungsprozess die MPC, basieren zunehmend auf nichtlinearen Prozessmodellen. Das Finden nichtlinearer Prozessmodelle ist notwendig, da viele technische Prozesse oftmals ein nichtlineares Verhalten aufweisen. Davon wird auch beim VGF-Kristallzüchtungsprozess

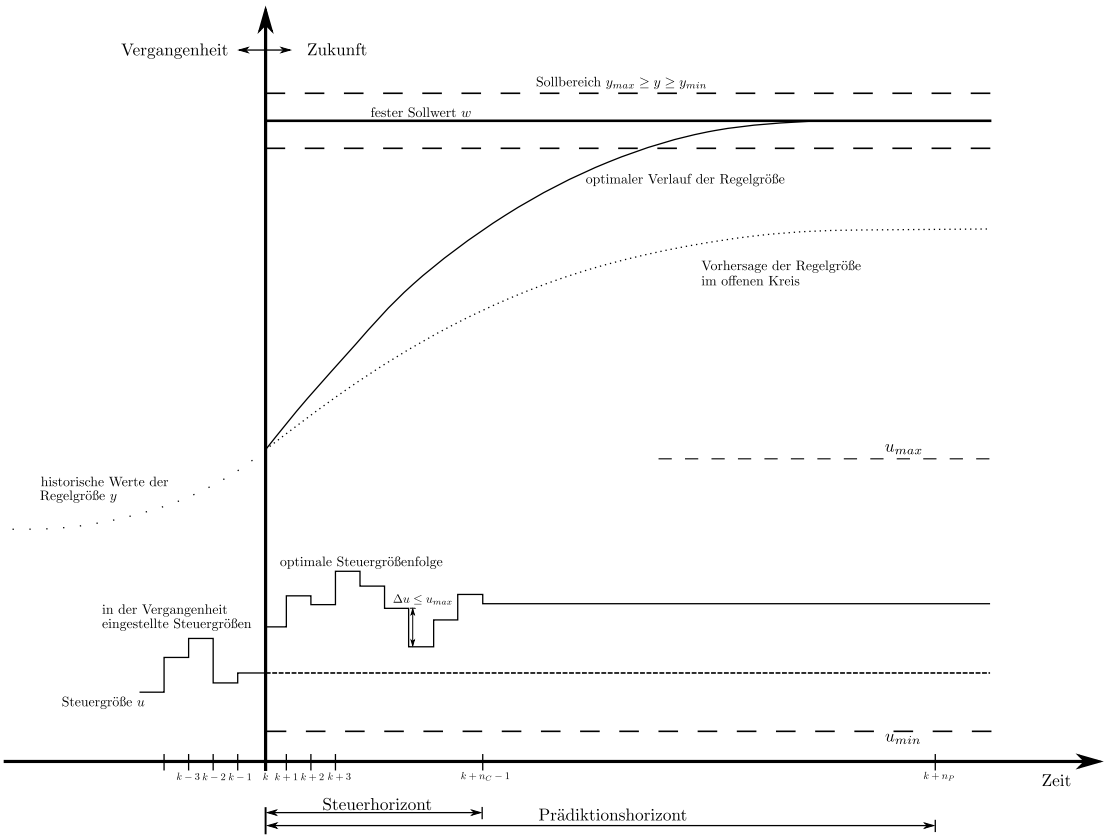

Abbildung 3 – Grundprinzip der modellprädiktiven Regelung [8]

ausgegangen. In dieser Arbeit sollen Systeme der Form

$$\frac{d}{dt}\boldsymbol{x}(t) = \dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{u}(t)) \tag{2.1}$$

betrachtet werden. Dabei ist $\boldsymbol{x}(t) \in \mathbb{R}^n$ der Zustand und $\boldsymbol{u}(t) \in \mathbb{R}^m$ der Eingang.[1] Die Funktion f beschreibt das dynamische Verhalten des Systems. Gleichung (2.1) ist ein nichtlineares System mit Zuständen und Eingängen. Um ein solches System zu identifizieren gibt es verschiedene Ansätze.

Die Identifikation linearer Systeme ist seit ca. 1960 Gegenstand der Forschung und weit vorangeschritten. Einschlägige Literatur hierzu sind zum Beispiel [16] und [13]. Auch die Identifikation nichtlinearer Systeme reicht bis in das letzte Jahrhundert zurück. In diesem Zusammenhang spielen KNN eine immer größer werdende Rolle. Insbesondere dynamische KNN können zur Identifikation des Ein-/Ausgangsverhaltens genutzt werden. Ein wichtiger Vertreter im Zusammenhang KNN mit der Systemidentifikation ist das rekurrente neuronale Netz [12]. Dabei kann jede Verbindung zwischen zwei Neuronen einen Zustand darstellen. Einen Überblick kurzen Überblick über die verschiedenen Arten von KNN im Zusammenhang mit nichtlinearer Systemidentifikation ist in [12] gegeben. In dieser Arbeit soll der Ansatz der KNN auch in einen Systemidentifikationsansatz münden. Dabei wird ein KNN mit einem Differentialgleichungslöser kombiniert [7].

Aufgrund der gestiegenen Rechenkapazitäten haben die numerischen Methoden in der nichtlineare Systemidentifikation eine wichtige Rolle erlangt. So kann die Finite-Elemente-Methode genutzt werden, um eine mathematische Berechnung des Temperaturverlaufs im Tiegel durchzuführen. Das führt zu einem nichtlinearen System mit einer hohen Zustandsdimension, die danach durch eine Ordnungsreduktion reduziert wird. Um eine solche numerische Berechnung durchzuführen, ist das physikalische und chemische Wissen der Prozesse im Tiegel unabdingbar [1]. Außerdem werden hohe Rechenkapazitäten benötigt. Deswegen soll sich in dieser Arbeit mit einem Ansatz beschäftigt werden, der weniger rechenintensiv ist und auf Grundlage der Informationen der Heizer angewendet werden kann [4].

2.3.1 Künstliche Neuronale Netze

Wie oben beschrieben, können zur Systemidentifikation KNN verwendet werden. Einer der ausgewählten Ansätze basiert auf der Nutzung eines KNN. Deswegen soll an dieser Stelle eine kurze Einführung in KNN gegeben werden.

Im maschinellen Lernen ist es üblich, die Datenbasis in verschiedene Datensets zu unterteilen. Bei neuronalen Netzen sind das

– Trainingsdaten,

[1] Im Folgenden wird auf die explizite Angabe der zeitlichen Abhängigkeit verzichtet.

- Validierungsdaten,
- Testdaten.

Zumeist ist die Aufteilung 70 % Trainingsdaten, 15 % Validierungsdaten und 15 % Testdaten. Falls keine Validierungsdaten gebildet werden, geht der Anteil zu den Testdaten über. Durch die Aufteilung in diese Datensets wird sichergestellt, dass neuronale Netze sich nicht nur an die gegebenen Daten anpassen, sondern generalisieren. Die Validierungsdaten verhindern eine Überanpassung des Netzes. In dieser Arbeit werden Trainings-, Validierungs- und Testdaten verwendet.

Entscheidend für den Erfolg des Ansatzes ist die Struktur des KNNes, dessen wichtigste Elemente im Zusammenhang mit Tensorflow die folgenden sind:

- Das Neuron: Das Neuron ist das Herzstück eines KNNes. Es ist dem biologischen Vorbild nachempfunden, allerdings unter Beibehaltung der wesentlichen Eigenschaften. Der Beginn künstlicher Neuronen war 1943 mit der McCulloch-Pits-Zelle, die jedoch nur binäre Signale verwenden kann. Dagegen besteht heute ein Neuron aus vier Basiselementen:
  - Gewichtung: Jeder Eingang χ_i eines Neurons j wird mit einem Gewicht w_{ij} multipliziert. Damit wird der Grad des Einflusses des Eingangs festgelegt. Die Gewichte können negativ (hemmend - inhibitorisch) oder positiv (erregend - exzatorisch) wirken. Sollte das $w_{ij} = 0$ sein, ist die Verbindung zwischen zwei Knoten nicht existent.
  - Übertragungsfunktion: Anhand der Eingaben berechnet die Übertragungsfunktion Σ die Netzeingabe υ des Neurons.
  - Aktivierungsfunktion: Die Ausgabe des Neurons wird durch die Aktivierungsfunktion φ bestimmt. Die Aktivierung wird durch die Netzeingabe und den Schwellenwert beeinflusst. Der Aktivierungsfunktion kommt eine besondere Bedeutung zu. Die Art des Problems, die das KNN lernen soll, ist maßgeblich für die Wahl der Aktivierungsfunktion. Übliche Aktivierungsfunktionen sind zum Beispiel die Schwellenwertfunktion oder die Sigmoidfunktion $\varphi = \frac{1}{1+e^{-a\upsilon}}$ mit dem variablen Steigungsmaß a. Nur durch eine nichtlineare Aktivierungsfunktion kann nichtlineares Verhalten des Netzeingangs auf den Netzausgang abgebildet werden. In dieser Arbeit werden hauptsächlich ein Tangens-Hyperbolicus $\varphi = \tanh(\upsilon)$ und rectified linear unit (ReLU) $\varphi = \max(0, \upsilon)$ als Aktivierungsfunktion für die Eingangsschicht und verborgenen Schichten verwendet. Bei der Ausgangsschicht wird in Systemidentifikationsnetzen standardmäßig eine lineare Aktivierungsfunktion benutzt. Es hat sich in Vergangenheit gezeigt, dass die Verwendung von ReLU erfolgreicher ist, als mit der Sigmoidfunktion [9]. Die Tangens-Hyperbolicus- und ReLU-Aktivierungsfunktion sind in Abbildung 4 dargestellt.

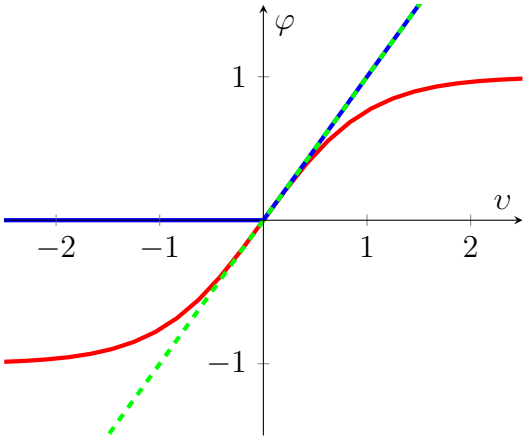

Abbildung 4 – Tangens-Hyperbolicus- (rot), ReLU- (blau) und lineare (grün) Aktivierungsfunktionen

- Schwellenwert: Durch die Addition eines Schwellenwertes Θ_j zur Netzeingabe υ, erfolgt eine Verschiebung der gewichteten Eingaben. Das biologische Äquivalent ist das Schwellenpotential bei Nervenzellen.

Mit diesen Elementen kann die Netzeingabe υ durch

$$\upsilon_j = \sum_{i=1}^{n} \chi_i w_{ij} \tag{2.2}$$

und die Aktivierung o_j durch

$$o_j = \varphi(\upsilon_j - \Theta_j) \tag{2.3}$$

definiert werden. In dieser Arbeit soll der Schwellenwert durch einen Weiteren Eingang x_0 (Bias) dargestellt werden. Damit ergeben sich die Gleichungen (2.2) und (2.3) zu

$$\upsilon_j = \sum_{i=0}^{n} \chi_i w_{ij} \tag{2.4}$$

bzw.

$$o_j = \varphi(\upsilon_j). \tag{2.5}$$

Der prinzipielle Aufbau eines Neurons ist in Abbildung 5 dargestellt.

- Die Schichten: Ein KNN besteht aus einer oder mehreren Schichten die wiederum aus einer unterschiedlichen Anzahl von Neuronen bestehen. Dabei gibt es in dem hier verwendeten KNN eine Eingangsschicht, mit $\dim(\boldsymbol{x}) + \dim(\boldsymbol{u})$ Neuronen, h verborgene Schichten mit o Neuronen und eine Ausgangsschicht mit $\dim(\boldsymbol{x})$ Neuronen. Sobald $h > 1$ ist, spricht man von einem tiefen neuronalen Netz. Die Schichten sind durch die Neuronen mit den oben bereits erwähnten Gewichten miteinander verknüpft. Wie die Schichten miteinander verknüpft sind, hängt von

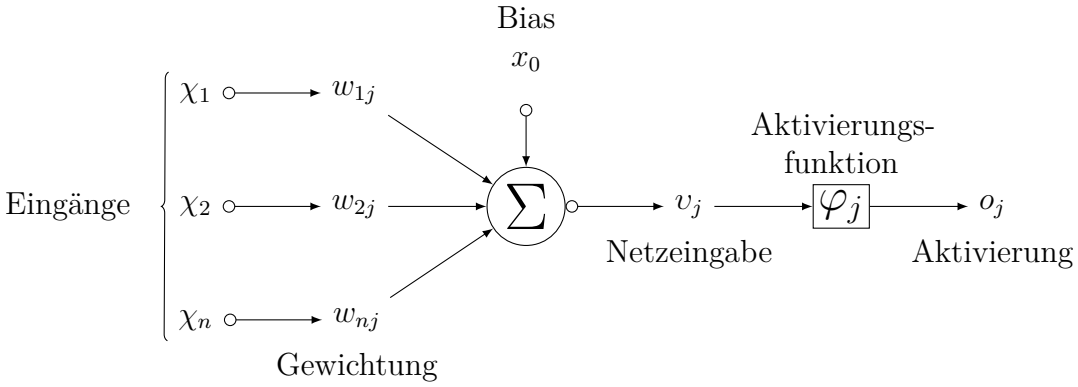

Abbildung 5 – Schematische Darstellung eines Neurons

der Art der Schichten ab. In dieser Arbeit werden insbesondere Schichten verwendet, bei denen jedes Neuron der Schicht i mit jedem Neuron der darauffolgenden Schicht $i+1$ verknüpft ist. Eine schematische Darstellung eines KNNes ist in Abbildung 6 gegeben.

Weitere erforderlich Elemente für das Funktionieren eines KNN sind:

- Die Kostenfunktion: Die Kostenfunktion ist die Funktion, die der Optimierer minimieren muss. Sie ist die Maßzahl zur Bestimmung der Abweichung zwischen der Prognose des Modells und den Referenzdaten. Da es sich hier um eine Regression handelt wird, wie üblich, die mittlere quadratische Abweichung (MQA) als Kostenfunktion angesetzt. Auf die MQA wird in Abschnitt 3.3 näher eingegangen.

- Der Optimierer: Der Optimierer berechnet auf Basis der Modellabweichung der Kostenfunktion die Gewichte und Biaswerte, also die Netzparameter θ, des KNNes während des Trainings. Ziel des Optimierers ist es, die Kostenfunktion zu minimieren und optimale Werte für die Gewichte und Biaswerte so zu finden, dass die Modellabweichung möglichst gering wird. Häufig verwendete Optimierer in Keras sind der *Adam*- und *RMSprop*-Optimierer. Diese Optimierer finden auch Anwendung in den hier durchgeführten Versuchen. Der Adam-Optimierer hat gegenüber dem RMSprop-Optimierer den Vorteil, dass er eine integrierte Lernratensteuerung enthält [15]. Damit kann die Lernrate am Anfang größer gewählt werden und diese wird dann automatisch verringert, um eine bessere Konvergenz zum Minimum zu erzielen.

Für eine ausführliche Einleitung in KNN wird auf [25] und [21] verwiesen.

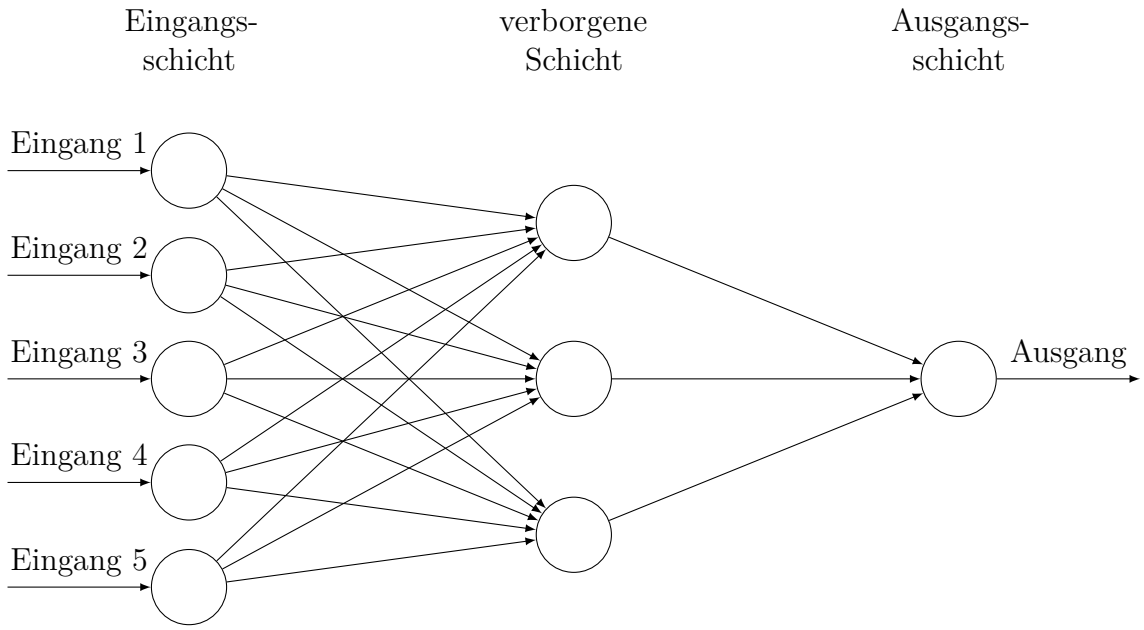

Abbildung 6 – Schematischer Aufbau eines KNN mit $h = 1$. Die Dimension des Netzeingangs ist $\dim \boldsymbol{x} + \dim \boldsymbol{u}$ und die Dimension des Ausgangs $\dim \boldsymbol{x}$.

2.3.2 Sparse identification of nonlinear dynamics with control

Der SINDYc-Ansatz basiert auf dem Lernen von Koeffizienten von Ansatzfunktionen. Ausschlaggebend ist, dass die Matrix, die die Koeffizienten enthält, dünn besetzt ist. Dadurch ist eine schnelle Berechnung des zukünftigen Verhaltens während der Regelung möglich. Das Verfahren wurde von Brunton, Proctor und Kutz in [3] vorgestellt und ist für lineare sowie nichtlineare Systeme geeignet. Ein entscheidender Vorteil dieses Ansatzes ist, dass zeitkontinuierliche Systeme berechnet werden können, auch wenn nur zeitdiskrete Messdaten vorliegen.

Wie in Abschnitt 2.3 vorgestellt, wird im SINDY-Ansatz von Systemen der Form (2.1) ausgegangen. Allerdings wird zuerst der Fall ohne Eingang $\boldsymbol{u}$ betrachtet. Die Diskussion des Ansatzes auf Eingänge erfolgt später in Abschnitt 2.3.2.

Der Ansatz basiert darauf, dass f nur aus einigen Termen besteht, also im Raum möglicher Funktionen dünn besetzt ist. Ziel ist es, das Verhalten des Systems, ohne Kenntnis der Funktion f, durch Minimierung einer Koeffizientenmatrix $\boldsymbol{\Xi}$ im Zusammenspiel mit einer Matrix Θ mit Ansatzfunktionen zu finden. Dann kann das System aus (2.1) durch

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}) \approx \boldsymbol{\Xi}^T(\boldsymbol{\Theta}(\boldsymbol{x}^T)) \tag{2.6}$$

dargestellt werden. Dafür wird der Zustand $\boldsymbol{x}$ für verschiedene Zeitpunkte $t_1, t_2, \ldots, t_m$

ermittelt und die Ableitung $\dot{\boldsymbol{x}} = \frac{d}{dt}\boldsymbol{x}$ numerisch ermittelt oder gemessen.

Für jeden Zustand wird eine Zeitfolge des Zustands der Form

$$\boldsymbol{x}_i(t) = x_i = \begin{pmatrix} x_i(t_1) \\ x_i(t_2) \\ \vdots \\ x_i(t_m) \end{pmatrix} \in \mathbb{R}^m \tag{2.7}$$

und eine abgeleitete Zeitfolge des Zustands

$$\dot{\boldsymbol{x}}_i(t) = \dot{\boldsymbol{x}}_i = \begin{pmatrix} \dot{x}_i(t_1) \\ \dot{x}_i(t_2) \\ \vdots \\ \dot{x}_i(t_m) \end{pmatrix} \in \mathbb{R}^m \tag{2.8}$$

gemessen bzw. berechnet. Für $i = 1, \ldots, n$ Zustände können diese Vektoren in den Matrizen

$$\boldsymbol{X} = \begin{pmatrix} \boldsymbol{x}_1 & \boldsymbol{x}_2 & \ldots & \boldsymbol{x}_n \end{pmatrix} = \begin{pmatrix} x_1(t_1) & x_2(t_1) & \ldots & x_n(t_1) \\ x_1(t_2) & x_2(t_2) & \ldots & x_n(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ x_1(t_m) & x_2(t_m) & \ldots & x_n(t_m) \end{pmatrix} \in \mathbb{R}^{m \times n} \tag{2.9}$$

bzw.

$$\dot{\boldsymbol{X}} = \begin{pmatrix} \dot{\boldsymbol{x}}_1 & \dot{\boldsymbol{x}}_2 & \ldots & \dot{\boldsymbol{x}}_n \end{pmatrix} = \begin{pmatrix} \dot{x}_1(t_1) & \dot{x}_2(t_1) & \ldots & \dot{x}_n(t_1) \\ \dot{x}_1(t_2) & \dot{x}_2(t_2) & \ldots & \dot{x}_n(t_2) \\ \vdots & \vdots & \ddots & \vdots \\ \dot{x}_1(t_m) & \dot{x}_2(t_m) & \ldots & \dot{x}_n(t_m) \end{pmatrix} \in \mathbb{R}^{m \times n}. \tag{2.10}$$

zusammengefasst werden.

Die Ansatzfunktionen können zum Beispiel Polynome höheren Grades oder trigonometrische Funktionen sein. Die Polynome lassen sich durch

$$p_k^\alpha = (x_1 + x_2 + \ldots + x_k)^\alpha = \sum_{\alpha_1 + \ldots + \alpha_k = \alpha} x_1^{\alpha_1} \cdot x_2^{\alpha_2} \ldots x_k^{\alpha_k} \tag{2.11}$$

bilden. Dabei ist α der Polynomgrad und k die Anzahl der Variablen. Für diese Arbeit wird (2.11) in Vektorschreibweise gebracht. Das ist mit $\alpha = 2$ und $k = 2$

$$\begin{aligned} p_2^2 &= x_1^2 + x_1 x_2 + x_2^2 \\ &= \underbrace{\begin{pmatrix} x_1^2 & x_1 x_2 & x_2^2 \end{pmatrix}}_{\boldsymbol{p}_2^2} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}. \end{aligned} \tag{2.12}$$

Dieser Vektor wird für alle Zweitpunkte t_i ausgewertet und auch wieder in Matrixform gebracht

$$\boldsymbol{P}_2^2 = \begin{pmatrix} x_1^2(t_1) & x_1(t_1)x_2(t_1) & x_2^2(t_1) \\ x_1^2(t_2) & x_1(t_2)x_2(t_2) & x_2^2(t_2) \\ \vdots & \vdots & \vdots \\ x_1^2(t_m) & x_1(t_m)x_2(t_m) & x_2^2(t_m) \end{pmatrix} \in \mathbb{R}^{m\times 3}. \tag{2.13}$$

Für k Variablen erweitert sich die Matrix dementsprechend. Allgemein gilt $\boldsymbol{P}_k^\alpha \in \mathbb{R}^{m\times \frac{(k+\alpha)!}{2k!\alpha!}}$.

Anschließend wird eine Bibliotheksmatrix $\boldsymbol{\Theta}(\boldsymbol{X})$ aus den Ansatzfunktionen generiert. Dabei werden die Polynommatrizen aneinandergereiht in die Form

$$\boldsymbol{\Theta}(\boldsymbol{X}) = \begin{pmatrix} \mathbf{1} & \boldsymbol{P}_k^1 & \boldsymbol{P}_k^2 & \dots & \boldsymbol{P}_k^\alpha & \sin(\boldsymbol{X}) & \cos(\boldsymbol{X}) & \dots \end{pmatrix} \in \mathbb{R}^{m\times l} \tag{2.14}$$

gebracht. Es können auch andere Funktionen, wie zum Beispiel $\sin(\boldsymbol{X})$ und $\cos(\boldsymbol{X})$ in die Bibliotheksmatrix geschrieben werden. Für $\sin(\boldsymbol{X})$ stellen diese sich wie folgt dar:

$$\begin{aligned} \sin(\boldsymbol{X}) &= \begin{pmatrix} \sin(\boldsymbol{x}_1) & \sin(\boldsymbol{x}_2) & \dots & \sin(\boldsymbol{x}_n) \end{pmatrix} \\ &= \begin{pmatrix} \sin(x_1(t_1)) & \sin(x_2(t_1)) & \dots & \sin(x_n(t_1)) \\ \sin(x_1(t_2)) & \sin(x_2(t_2)) & \dots & \sin(x_n(t_2)) \\ \vdots & \vdots & \ddots & \vdots \\ \sin(x_1(t_m)) & \sin(x_2(t_m)) & \dots & \sin(x_n(t_m)) \end{pmatrix} \in \mathbb{R}^{m\times n}. \end{aligned} \tag{2.15}$$

Jetzt kann mit Einführung einer Koeffizientenmatrix $\boldsymbol{\Xi} = \begin{pmatrix} \boldsymbol{\xi}_1 & \boldsymbol{\xi}_2 & \dots & \boldsymbol{\xi}_n \end{pmatrix} \in \mathbb{R}^{l\times n}$ das Regressionsproblem

$$\dot{\boldsymbol{X}} = \boldsymbol{\Theta}(\boldsymbol{X})\boldsymbol{\Xi} \tag{2.16}$$

erstellt werden. Dieses Problem ist dünn besetzt, weil angenommen wird, dass nur wenige der Ansatzfunktionen in $\boldsymbol{\Theta}(\boldsymbol{X})$ aktiv sind. Demzufolge werden auch die Koeffizientenvektoren $\boldsymbol{\xi}_1, \boldsymbol{\xi}_2, \dots, \boldsymbol{\xi}_n$ dünn besetzt sein. Dies wird in Abschnitt 3.3 gezeigt.

Die Koeffizienten können durch ein Optimierungsproblem gefunden werden. Brunton, Proctor und Kutz nutzen dafür die least absolute shrinkage and selection operator (LASSO)-Regression. Es sind aber auch andere Minimierungsansätze möglich. LASSO ist eine lineare Regression, die speziell bei dünn besetzten Minimierungsproblemen vorteilhaft ist und gleichzeitig eine Schrumpfung und Variablenselektion durchführt.

Es wird angenommen, dass das lineare Modell

$$z_i = \beta_0 + \beta_1 v_{i,1} + \beta_2 v_{i,2} + \dots + \beta_p v_{i,q} + \epsilon_i \tag{2.17}$$

mit der Zielgröße z_i, den Modellvariablen $v_{i,1}, v_{i,2}, \dots, v_{i,q}$ und den Modellkoeffizienten $\beta_0, \beta_1, \dots, \beta_q$ geschätzt werden soll. Der zu minimierende Modellfehler ist ϵ_i. Dieser Zusammenhang kann in Vektorschreibweise durch

$$\underbrace{\boldsymbol{z}}_{\in\mathbb{R}^m} = \underbrace{\boldsymbol{V}}_{\in\mathbb{R}^{m\times q}} \underbrace{\boldsymbol{\beta}}_{\in\mathbb{R}^q} + \underbrace{\boldsymbol{\epsilon}}_{\in\mathbb{R}^m} \tag{2.18}$$

geschrieben werden. Der kleinste quadrate Schätzer kann durch

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \|\underbrace{\boldsymbol{V}\boldsymbol{\beta} - \boldsymbol{z}}_{=\boldsymbol{\epsilon}}\|_2^2 \tag{2.19}$$

berechnet werden. Dabei ist $\|\cdot\|_2$ die euklidische Norm und $\hat{\boldsymbol{\beta}} \in \mathbb{R}^q$ der Vektor der geschätzten Koeffizienten [26].

LASSO wird von Tibshirani in [27] eingeführt und folgendermaßen dargestellt:

$$\begin{aligned} \hat{\boldsymbol{\beta}} = \arg\min & \left\{ \sum_{i=1}^{N} \left(z_i - \beta_0 - \sum_{j=1}^{q} \beta_j v_{ij} \right)^2 \right\} \\ & \text{N.B.:} \sum_j |\beta_j| \leq \lambda. \end{aligned} \tag{2.20}$$

Die Nebenbedingung in (2.20) führt eine 1-Norm-Penalisierung während der Minimierung durch. Dieser Term wird auch Strafterm genannt. Mit steigender Summe der Absolutbeträge von β_j, wird dieser immer größer. Dadurch werden, je nach Wahl von λ, die Koeffizienten geschrumpft, bzw. einige exakt auf Null gesetzt.

Mit Integration der Nebenbedingung in das Minimierungsproblem (2.19) kann der LASSO-Ansatz beim SINDY-Ansatz mittels

$$\boldsymbol{\xi}_k = \arg\min_{\boldsymbol{\xi}_k} (\|\dot{\boldsymbol{X}} - \boldsymbol{\xi}_k \boldsymbol{\Theta}^T(\boldsymbol{X})\|_2^2 + \lambda \|\boldsymbol{\xi}_k\|_1), \tag{2.21}$$

mit $\lambda \geq 0$, ausgedrückt werden.

Nach Berechnung der Koeffizienten können die Systemgleichungen mittels Gleichung (2.6) geschrieben werden.

Sparse identification of nonlinear dynamics with control

Um den SINDYc-Ansatz der Systemidentifikation für die Regelungstechnik sinnvoll nutzbar zu machen, ist ein Eingang $\boldsymbol{u}$ unabdingbar. Dafür ist eine Modifizierung des Ansatzes nötig, weswegen dieser Sparse identification of nonlinear dynamics *with control* genannt wird [4].

Ziel ist es Systeme wie in (2.1) eingeführt zu identifizieren. Demzufolge muss die Matrix $\boldsymbol{\Theta}(\boldsymbol{x}, \boldsymbol{u})$ auch Ansatzfunktionen für den Eingang $\boldsymbol{u}$ enthalten. Diese Funktionen können auch nichtlineare gemischte Terme aus $\boldsymbol{x}$ und $\boldsymbol{u}$ enthalten. Die Integration der Terme in die Matrix Ξ erfolgt analog zu den oben erklärten Abläufen. Dabei erhöht sich auch die Dimension der Koeffizientenmatrix $\boldsymbol{\Xi}$. Dann müssen neben den Zuständen $\boldsymbol{x}$ auch die Eingänge $\boldsymbol{u}$ messbar sein. Die Eingangsdaten werden analog zu den Zuständen in Matrix $\boldsymbol{\Upsilon}$ geschrieben.

Es ergibt sich damit der Ansatz äquivalent zu (2.16) zu

$$\dot{\boldsymbol{X}} = \boldsymbol{\Xi}\boldsymbol{\Theta}^T(\boldsymbol{X}, \boldsymbol{\Upsilon}). \tag{2.22}$$

Dieser Ansatz setzt voraus, dass $\boldsymbol{u}$ ein Eingang ist. Ist dies nicht der Fall, und handelt es sich dabei um ein zurückgeführtes Signal, funktioniert dieser Ansatz nicht. Hierbei wird auf die Lösung von Brunton, Proctor und Kutz in [4] verwiesen, weil das nicht Gegenstand der Arbeit ist.

2.3.3 Ordinary differential equation network

Der zweite Ansatz, der zur Systemidentifikation gewählt wird, basiert auf den KNN. Hierbei stütze ich mich auf die Arbeit von Chen u. a. in [7].

Das Grundprinzip ist, nicht wie bei anderen Systemidentifikationsansätzen mit KNN, die Lösung der Differentialgleichung (DGL) zu lernen. Stattdessen wird nur die Systemdynamik gelernt und das Lösen der DGL erfolgt duurch einen Differentialgleichungslöser. Das hat den Vorteil, dass das KNN nicht die Aufgabe eines Differentialgleichungslösers übernehmen muss, sondern ein dafür optimierter Löser diese Aufgabe ausführt. Die Erwartung ist, dass damit gute Systemidentifikationsergebnisse erzielt werden.

Wie in Abschnitt 2.3 ausgeführt, findet in der Systemidentifikation oftmals ein rekurenntes neuronales Netz (RNN) Anwendung. Solche KNN beinhalten sich wiederholende Blöcke aus Schichten, die es ermöglichen, sequentielle Informationen zu speichern und in jedem Schritt durch eine gelernte Funktion zu ändern. Solche KNN können durch die Gleichung

$$\boldsymbol{h}_{t+1} = \boldsymbol{h}_t + f(\boldsymbol{h}_t, \theta_t) \tag{2.23}$$

beschrieben werden. Dabei ist $\boldsymbol{h}_t \in \mathbb{R}^D$ die „verborgene“ Information zum Zeitpunkt $t = 0 \dots t_{max}$ und $f(\boldsymbol{h}_t, \theta_t)$. Die gelernte Funktion der verborgenen Information mit den Parametern θ. Gleichung (2.23) weißt auffallende Ähnlichkeit mit dem expliziten Euler-Verfahren zum numerischen Lösen von DGL auf. Beim expliziten Euler-Verfahren kann die Lösung der DGL

$$\dot{y} = f(t, y), y(t_0) = y_0 \tag{2.24}$$

durch Betrachtung an den diskreten Zeitpunkten $t_k = t_0 + kh, k = 0, 1, 2, \dots$, mittels

$$y_{k+1} = y_k + hf(t_k, y_k) \tag{2.25}$$

berechnet werden, wobei $h > 0$ die Schrittweite ist.

Chen u. a. führen nun mehr Schichten und kleinere Schritte ein. Dies wäre äquivalent zu einer Erhöhung der Berechnungen in einem RNN. Für eine Reduzierung der Schrittweite Δt gegen 0, kann (2.23) als DGL geschrieben werden:

$$\frac{d\boldsymbol{h}(t)}{dt} = f(\boldsymbol{h}(t), t, \theta). \tag{2.26}$$

Durch Lösen dieser DGL, kann die gewünschte Abfolge verborgener Zustände $\boldsymbol{h}(t)$ berechnet werden. Hierbei handelt es sich um ein Anfangswertproblem mit dem Anfangswert $\boldsymbol{h}(0)$. Dieser Anfangszustand lässt sich als Eingangsschicht eines RNNes darstellen.

Die Ausgangsschicht ist demzufolge $\boldsymbol{h}(t_{max})$. Der Differentialgleichungslöser muss nun die verborgene Dynamik f auswerten, um $\boldsymbol{h}(t_{max})$ zu berechnen. Die Dynamik f soll dabei von einem KNN gelernt werden.

Die numerische Lösung von DGL erfolgt oftmals durch Integration. Die meisten Methoden sind allerdings sehr zeitintensiv. Dies ist insbesondere beim Training der KNN hinderlich, weil dabei die Integrationsschritte differenziert werden müssen um die Netzwerkparameter θ zu optimieren (auch als Fehlerrückführung oder Rückpropagierung bezeichnet). Das hätte einen sehr hohen Rechenbedarf zur Folge.

Um dieses Problem zu umgehen, nutzen Chen u. a. die von Pontryagin vorgestellte *adjunkten Methode* [23]. Dabei wird eine zweite DGL rückwärts gelöst. Diese Methode kann mit allen Differentialgleichungslösern, unter geringer Rechenleistung, verwendet werden. Dafür wird die Annahme getroffen, dass ein Kostenfunktional, das das Ergebnis eines Differentialgleichungslösers minimiert, durch

$$L(\boldsymbol{h}(t_1)) = L\left(\boldsymbol{h}(t_0) + \int_{t_0}^{t_1} f(\boldsymbol{h}(t), t, \theta)dt\right) = L(\text{ODESolve}(\boldsymbol{h}(t_0), f, t_0, t_1, \theta)) \quad (2.27)$$

geschrieben werden kann. Um L zu optimieren werden Ableitungen nach θ benötigt. Dafür wird im ersten Schritt evaluiert, wie der Gradient von L vom verborgenen Zustand $\boldsymbol{h}(t)$ abhängt. Dies kann durch die Adjunkte

$$\boldsymbol{a}(t) = \frac{\partial L}{\partial \boldsymbol{h}(t)} \quad (2.28)$$

ausgedrückt werden. Chen u. a. nehmen dabei an, dass $\boldsymbol{a}(t)$ eine orthogonale Matrix ist. Die Dynamik davon ist wiederum als DGL mittels Anwenden der Kettenregel durch

$$\frac{d\boldsymbol{a}(t)}{dt} = -\frac{\partial \boldsymbol{h}(t)}{\partial L}\frac{\partial f(\boldsymbol{h}(t), t, \theta)}{\partial \boldsymbol{h}} = -\boldsymbol{a}(t)^T\frac{\partial f(\boldsymbol{h}(t), t, \theta)}{\partial \boldsymbol{h}} \quad (2.29)$$

gegeben. $\frac{\partial L}{\partial \boldsymbol{h}(t_0)}$ kann durch Rückwärtsrechnen eines Differentialgleichungslösers gefunden werden, mit dem Anfangswert $\frac{\partial L}{\partial \boldsymbol{h}(t_1)}$. Um diese DGL zu lösen muss $\boldsymbol{h}(t)$ an der gesamten Trajektorie bekannt sein. $\boldsymbol{h}(t)$ kann rückwärts mit der Adjunkten mit dem Startwert $\boldsymbol{h}(t_1)$ berechnet werden. Um jetzt die Ableitungen nach den Parametern θ zu berechnen muss noch das dritte Integral

$$\frac{dL}{d\theta} = \int_{t_1}^{t_0} \boldsymbol{a}(t)^T\frac{\partial f(\boldsymbol{h}(t), t, \theta)}{\partial \theta}dt \quad (2.30)$$

ausgerechnet werden.

Wie dem Algorithmus 1 zu entnehmen ist, wird zuerst ein Zustand $s_0 = [\boldsymbol{h}(t_1), \frac{\partial L}{\partial \boldsymbol{h}(t_1)}, \mathbf{0}_{|\theta|}]$ erstellt. Daraufhin werden die Dynamik f des Zustandes definiert und die nötigen Berechnungen vorgenommen. Zuletzt wird die DGL rückwärts gelöst und die Gradienten zurückgegeben.

Algorithmus 1: Rückwärtsdifferenzierung eines DGL-Anfangswertproblems

Eingang: Dynamik, Parameter θ, Startzeit t_0, Endzeit t_1, Endzustand $\boldsymbol{h}(t_1)$, Ableitung $\frac{\partial L}{\partial \boldsymbol{h}(t_1)}$

$s_0 = [\boldsymbol{h}(t_1), \frac{\partial L}{\partial \boldsymbol{h}(t_1)}, \mathbf{0}_{|\theta|}]$

def *aug_dynamics*$([\boldsymbol{h}(t), \boldsymbol{a}(t), \cdot], t, \theta)$**:**

  zurück $[f(\boldsymbol{h}(t), t, \theta), -\boldsymbol{a}(t)^T \frac{\partial f}{\partial \theta}]$

$[\boldsymbol{h}(t_0), \frac{\partial L}{\partial \boldsymbol{h}(t_0)}, \frac{\partial L}{\partial \theta}] = \text{ODESolve}(s_0, \text{aug\_dynamics}, t_1, t_0, \theta)$

zurück $\frac{\partial L}{\partial \boldsymbol{h}(t_0)}, \frac{\partial L}{\partial \theta}$

2.3.4 Langes Kurzzeitgedächtnis Künstliches Neuronales Netz

Eine Methode mit KNN, die in den letzten Jahren stark weiterentwickelt wurde, ist die Nutzung eines langes Kurzzeitgedächtnis (LSTM)-KNN. Ansätze zur Systemidentifikation wurden in [29] und [10] vorgestellt und erfolgreich praktiziert.

Die Vorteil der Nutzung eines LSTM-KNN gegenüber KNN liegt darin, dass das zur Optimierung der Gewichte genutzte Fehlerabstiegsverfahren verändert wird. Bei tiefen KNN kann das Fehlerabstiegsverfahren durch die vielen Schichten in einem lokalen Minimum enden und nicht wie eigentlich gewollt, im globalen Minimum. Deswegen erfolgt bei tiefen KNN keine optimale Anpassung der Gewichte. Zur Lösung dieses Problems haben Hochreiter und Schmidhuber 1997 in [11] die LSTM-KNN vorgestellt. Dafür werden in einer LSTM-Zelle drei Torsorten verwendet: Ein Eingangstor, ein Ausgangstor und und Merk- und Vergesstor. Die Struktur einer LSTM-Zelle erlaubt einen konstanten Fehlerfluss. Dabei steuert das Eingangstor das Ausmaß, in dem ein neuer Wert in die Zelle fließt, das Merk- und Vergesstor das Ausmaß, in dem ein Wert in der Zelle verbleibt oder vergessen wird und das Ausgangstor das Ausmaß, in dem der Wert in der Zelle zur Berechnung für das nächste Modul des KNN verwendet wird [11]. Dieses Verfahren erlaubt die Steuerung der Fehlerrückführung und damit eine bessere Anpassung der Gewichte. Letztendlich resultiert das in einer Verbesserung der Systemidentifikation.

Die Implementierung eines LSTM-KNN erfolgte nur anfänglich, weil die beiden Ansätze SINDYc und ODEnet vielversprechender erschienen.

2.3.5 Komprimierte Erfassung

Die Idee, komprimierte Erfassung zur Identifikation nichtlinearer Systeme zu nutzen, stammt von Wang u. a. in [28] zur Prädiktion von Katastrophen. Auch in [20] wird die komprimierte Erfassung erfolgreich zur Identifikation nichtlinearer Systeme genutzt.

Die Grundidee der Systemidentifikation mit der komprimierten Erfassung ist äquivalent

zu der Annahme des SINDYc-Ansatzes. Ein dynamisches System kann durch nur wenige Funktionen mit entsprechenden Koeffizienten beschrieben werden. Die Schwierigkeit liegt darin, die entsprechenden Koeffizienten für die Funktionen zu finden. Wang u. a. nutzen die Annahme zur Identifikation von Systemen, bei denen auch die Datenbasis dünn besetzt ist. Darin liegt der Unterschied zum SINDYc-Ansatz, bei dem die Datenbasis nicht dünn besetzt ist. Die komprimierte Erfassung erweitert den SINDYc-Ansatz im Prinzip um die Rekonstruktion des Systems aus dünn besetzten Signalen.

Dieser Ansatz wurde in der Arbeit nicht implementiert, weil vorerst die Annahme getroffen wird, dass die Datenbasis nicht dünn besetzt ist. Deswegen soll auch nicht weiter auf die Hintergründe der komprimierten Erfassung eingegangen werden.

Kapitel 3

Implementierung der Ansätze und Durchführung von Tests

Um die ausgewählten Systemidentifikationsansätze auf die Tauglichkeit in der Regelungstechnik und in Verbindung mit dem VGF-Kristallzüchtungsprozess zu testen, ist eine Implementierung der Ansätze und von Testsystemen nötig. Anhand dieser kann schon im Voraus, ohne Wissen über die eigentlichen Messdaten des VGF-Kristallzüchtungsprozess, eine Abschätzung über die Eignung, Parameterabhängigkeiten und Rechenzeiten getroffen werden.

3.1 Wahl und Vorstellung der Testsysteme

Die Wahl der Testsysteme soll so erfolgen, dass die ausgewählten Identifikationsalgorithmen anhand dieser ausgiebig getestet werden können. Anfänglich soll mit diesen die Implementierung zu verifiziert und daran anschließend die Ergebnisqualität und Abhängigkeit von verschiedenen Faktoren beurteilt werden. Dabei werden die Systeme nicht willkürlich erstellt, sondern orientieren oder gleichen sich den Systemen, die auch in der Literatur verwendet werden. Damit ist auch ein Prüfen mit den Ergebnissen aus der Literatur möglich.

Zum Testen des SINDYc-Algorithmus werden die folgenden Systeme verwendet:

1. nichtlinearer Oszillators zweiter Ordnung mit und ohne Eingang,
2. die Lotka-Volterra-Gleichungen mit Eingang und
3. der Lorenz-Attraktor.

Alle drei Systeme werden auch in der Literatur verwendet [3], [4], [5].

Beispiel 1. *Nichtlinearer Oszillator zweiter Ordnung*

Der nichtlineare Oszillator kann als System zweier DGL mit den Zuständen x_1, x_2 und dem Eingang als

$$\begin{aligned} \dot{x}_1 &= ax_1^3 + bx_2^3 + u \\ \dot{x}_2 &= cx_1^3 + dx_2^3 \end{aligned} \tag{3.1}$$

dargestellt werden, wobei $u(t) \in \mathbb{R}$ gilt. Mit SINDYc müssen die Koeffizienten $a, b, c, d \in \mathbb{R}$ gefunden werden. Das System kann auch ohne den Eingang betrachtet werden. In diesem Fall muss auch kein SINDYc, sondern nur SINDY benutzt werden.

Das System wird als relativ einfach zu identifizierend eingestuft, weil nur vier Koeffizienten zu identifizieren sind und es keinerlei Mischterme aus x_1 und x_2 gibt. Es werden zur Identifikation des Systems demzufolge auch nur Polynomfunktionen bis zum dritten Grad benötigt.

Für die Tests der Ansätze werden die Parameter $a = -0{,}1, b = 2, c = -2, d = -0{,}1$ gewählt. Die benutzten Eingänge sind $u = t$ und $u = 2\sin t + 2\sin \frac{t}{10}$.

Beispiel 2. *Lotka-Volterra-Gleichungen*

Ein weiteres Testsystem, das in der der Literatur im Zusammenhang mit SINDYc oftmals verwendet wird, sind die die Lotka-Volterra-Gleichungen, die auch als Räuber-Beute-Gleichungen bezeichnet werden. Die gesteigerte Schwierigkeit im Gegensatz zu dem Beispiel 1 ergibt sich durch die Mischterme zwischen den einzelnen Zuständen. Die folgenden Gleichungen wurden von Alfred J. Lotka und Vito Volterra unabhängig voneinander in den Jahren 1925 bzw. 1926 aufgestellt [18].

$$\begin{aligned} \dot{x}_1 &= x_1(\rho_1 - \gamma_1 x_2) + u \quad \text{(Räuber)} \\ \dot{x}_2 &= -x_2(\rho_2 - \gamma_2 x_1) \quad \text{(Beute)} \end{aligned} \tag{3.2}$$

Der Eingang $u(t) \in \mathbb{R}$ ist im ursprünglichen System nicht vorgesehen und wird hier nur zu Testzwecken eingeführt. Es gilt hier $\rho_1, \rho_2, \gamma_1, \gamma_2 > 0$ [5]. Der Eingang simuliert einen äußeren Einfluss auf die Räuber. Es ist auch möglich, einen zweiten Eingang als Einfluss auf die Beute zu integrieren.

Für die Tests der Ansätze werden die Parameter $\rho_1 = 0{,}5, \gamma_1 = 0{,}025, \rho_2 = 0{,}5, \gamma_2 = 0{,}005$ gewählt. Die benutzten Eingänge sind $u = t$ und $u = 2\sin t + 2\sin \frac{t}{10}$.

Beispiel 3. *Lorenz-Attraktor*

Als drittes Testsystem wurde der Lorenz-Attraktor ausgewählt. Auch dieser wird in der

Literatur als Beispielsystem für SINDY und SINDYc verwendet [4], [5]. Der Lorenz-Attraktor ist ein System dreier gekoppelter, nichtlinearer gewöhnlicher DGL [17]:

$$\begin{aligned} \dot{x}_1 &= a(x_2 - x_1) + u \\ \dot{x}_2 &= x_1(b - x_3) - x_2 \\ \dot{x}_3 &= x_1 x_2 - c x_3 \end{aligned} \tag{3.3}$$

Erneut ist der Eingang u nur zu Testzwecken hinzugefügt. $a, b, c \in \mathbb{R}$ sind die Parameter des Lorenz-Attraktors. Die Bereich für a, b, c, für die der Lorenz-Attraktor identifizierbare Ergebnisse liefert, sind sehr eingeschränkt. Deswegen werden diese auf $a = 10, b = 28, c = 8/3$ festgelegt. Damit zeigt der Attraktor chaotisches Verhalten. Es ist interessant herauszufinden, ob SINDYc die Parameter aus dem chaotischen Verhalten identifizieren kann. Da der Verlauf der Messdaten des VGF-Kristallzüchtungsprozess zum Testzeitpunkt unbekannt ist, kann anhand dieses Tests möglicherweise eine Aussage darüber getroffen werden, inwieweit der Algorithmus chaotische Messdaten zu Identifikation nutzen kann.

Da für den ODEnet-Ansatz in der Literatur nur sehr spärlich Testsystem angegeben sind (beispielsweise ein linearer gedämpfter Oszillator in [7]), und die Systemidentifikation in dieser Arbeit auf jeden Fall nichtlinear ist, werden die Beispiele 1 und 2 zum Test verwendet. Damit kann auch eine Vergleichbarkeit der beiden Ansätze hergestellt werden.

3.2 Implementierung und Durchführung der Tests

Die Implementierung der ausgewählten Ansätze und der Testsysteme wird ausschließlich in *Python* realisiert. Dabei wird die Version 3.7 64 bit verwendet. Dies ist erforderlich um das ODEnet zu implementieren. Python bietet den Vorteil, dass es eine einfache, leicht zu erlernende und verstehende Syntax hat und durch Bibliotheken gut erweiterbar ist. Des Weiteren erfreut sich Python, insbesondere im Bereich des maschinellen Lernens, großer Beliebtheit. Der verwendete Computer ist ein Hewlett-Packard ProDesk mit Intel Core i3-4130, 8 GB RAM und Windows 8.1 64 bit.

3.2.1 SINDYc

Bei der Implementierung von SINDYc stellten sich hauptsächlich drei größere Probleme:

- der Optimierer, der die Minimierung durchführt,
- die Standardisierung der Daten und
- die Integration der dünnen Besetztheit.

In [4] wird als Regressionsalgorithmus die LASSO-Regression vorgeschlagen. Dies ist ein Regressionsalgorithmus, der 1996 von Tibshirani in [27] beschrieben wurde. Die Besonderheit ist, dass Variablenwahl und Regularisierung im LASSO durchgeführt werden. Dadurch werden nur die relevanten Koeffizienten in den Vektoren $\boldsymbol{\xi}_k$ als nicht null berechnet. Die Verwendung der LASSO-Regression ist ein Baustein zur Implementierung der dünnen Besetztheit.

Im Gebiet des maschinellen Lernens werden die Eingangsdaten in den Algorithmus in ein bestimmtes Format gebracht, weil sonst die Regularisierung (wie sie auch bei der LASSO-Regression verwendet wird) nicht richtig durchgeführt werden kann. Die Standardisierung erfolgt in der vorliegenden Implementierung mit der Python-Bibliothek *sklearn*, die eine Skalierungsklasse *StandardScaler* zur Verfügung stellt [22]. Dabei werden die Daten um den Mittelwert bereinigt und auf Einheitsvarianz skaliert. Die neue erhaltenen Daten können also durch

$$z_s = \frac{x - \mu}{s_s} \tag{3.4}$$

beschrieben werden, wobei μ der Mittelwert der Daten und s_s die Standardabweichung der Daten ist und x die unskalierten und z die skalierten Daten sind. Nach dem Trainieren werden die Daten wieder zurückgerechnet. Dazu wird (3.4) zu

$$x = z_s s_s + \mu \tag{3.5}$$

umgeformt. Dieses Vorgehen wird auch auf den Eingang angewendet.

Auch die Zeit wird skaliert. Dies erfolgt erneut durch die Benutzung der gleichen Bibliothek, allerdings wird hierfür ein *MinMaxScaler* verwendet. Dieser skaliert die Zeit $t = [t_{min}, t_{max}] \in \mathbb{R}$ durch

$$t_s = \frac{t - t_{min}}{t_{max} - t_{min}} \tag{3.6}$$

auf den Bereich $t_s = [0, 1] \in \mathbb{R}$ [22]. Durch die Skalierung ist ein besserer Vergleich verschiedener Systeme möglich. Das Zurückrechnen erfolgt erneut durch umstellen von (3.6) nach

$$t = t_s(t_{max} - t_{min}) + t_{min}. \tag{3.7}$$

Wie im Abschnitt 2.3.2 erläutert, sollen die Koeffizientenvektoren $\boldsymbol{\xi}_k$ für die Ansatzfunktionen dünn besetzt sein. Damit das erreicht wird und im Optimalfall nur sehr wenige Koeffizienten nicht null sind, müssen in der Implementierung verschiedene Methoden zusammenwirken. Die LASSO-Regression wird automatisch zu einer dünnen Besetztheit der Vektoren führen. Durch Beobachtungen an den Testsystemen konnte allerdings festgestellt werden, dass das nicht ausreicht, da λ für jedes System händisch eingestellt werden muss und dies sehr aufwendig ist. Es gäbe auch noch die Möglichkeit λ automatisch einzustellen, zum Beispiel durch die Anwendung des Pareto-Optimums. Da dies jedoch mit den genutzten Python-Funktionen sehr kompliziert und die Handeinstellungsmöglichkeiten einfach sind, wird in dieser Arbeit darauf verzichtet.

Das Training erfolgt in mehreren Iterationen. Sobald während einer Iteration eine Zeile in der Koeffizientenmatrix $\boldsymbol{\Xi}$ 0 ist, wird diese aus der Koeffizientenmatrix $\boldsymbol{\Xi}$ und die korrespondierenden Spalte in der Matrix $\boldsymbol{\Theta}$ gelöscht. Die Entscheidung, ob ein Koeffizient 0 ist, basiert auf einem Potenzunterschied ΔD. Dieser berechnet sich aus dem größten Koeffizienten eines jeden Koeffizientenvektors mit einem manuell, für das jeweilige System angepasstem, Faktor D, folgendermaßen:

$$\Delta D_k = \frac{\|\boldsymbol{\xi}_k\|_\infty}{D} \tag{3.8}$$

Falls ein Element im Vektor $\boldsymbol{\xi}_k$ kleiner ist als ΔD_k, wird dieses zu Null gesetzt. Es gibt demnach zwei miteinander korrespondierende Verfahren, um die dünne Besetztheit sicherzustellen: Die Nutzung der LASSO-Regression mit der Einstellung von λ und die Nutzung des Dimensionsunterschieds ΔD.

Im Beispiel 1 werden für die Datengenerierung die Parameter $a = -0{,}1, b = 2, c = -2, d = -0{,}1$. Gleichung (3.1) ergibt sich nach Einsetzen der Parameter zu

$$\begin{aligned} \dot{x}_1 &= -0,1x_1^3 + 2x_2^3 + u \\ \dot{x}_2 &= -2x_1^3 - 0,1x_2^3 \end{aligned} \tag{3.9}$$

Die Datengenerierung erfolgt für $t = [0, 25]$s mit einer Abtastzeit von $\Delta t = 0{,}01\,\mathrm{s}$. Außerdem wird bei den Tests mit Eingang der Eingang $u = t$ benutzt. Die generierten Daten werden, wie oben beschrieben, standardisiert und in den Matrizen $\boldsymbol{X}$ bzw. abgeleitet in $\dot{\boldsymbol{X}}$ gespeichert. Darauf erfolgt das Festlegen auf einen Polynomgrad, in diesem Fall $\alpha = 3$ und das Erstellen der Matrix $\boldsymbol{\Theta}$ mit den Ansatzfunktionen. Sobald dies erfolgt ist, kann die Minimierung nach 2.21 mit dem LASSO-Ansatz durchgeführt und die Koeffizientenmatrix $\boldsymbol{\Xi}$ mit den darin enthaltenen Koeffizientenvektoren $\boldsymbol{\xi}_k$ berechnet werden. Die Daten werden auch für die Systemidentifikation mit dem ODEnet benutzt.

Analog zu diesem Schema werden auch die Beispiele 2 und 3 implementiert. Bei beiden Systemen ist der Eingang jedoch durch $u = 2\sin t + 2\sin\frac{t}{10}$ gegeben. Zusätzlich wird bei Beispiel 3 eine Abtastzeit $\Delta t = 0{,}001\,\mathrm{s}$ verwendet. Die Erklärung für diese Maßnahme erfolgt im Abschnitt 3.3.3.

3.2.2 ODEnet

Für die Implementierung des ODEnet-Ansatzes wurde die Python-Bibliothek *TensorFlow* als Backend benutzt. Diese ist neben *PyTorch* die wohl bekannteste Bibliothek, um in Verbindung mit *Keras* als Frontend KNN zu erstellen und trainieren. Bei TensorFlow wird die neue Architektur 2.0 benutzt, weil diese 64 bit fähig ist und den sogenannten *Eager-Execution*-Modus hat, der Operationen direkt ausführt, ohne einen Graph zu erstellen, der später berechnet wird. Dadurch ist die Fehlersuche und -behebung vereinfacht. Die Implementierung der adjunkten Methode erfolgt in dieser Arbeit nicht, weil der Aufwand

Schicht	Art	Anzahl Neuronen	Aktivierungsfunktion
1	Eingangssicht	2 bzw. 3	keine
2	dichte verdeckte Schicht	50	ReLU
3	dichte verdeckte Schicht	50	ReLU
4	dichte verdeckte Schicht	50	ReLU
5	dichte verdeckte Schicht	50	ReLU
6	dichte verdeckte Schicht	50	ReLU
7	dichte Ausgangsschicht	2	linear

Tabelle 1 – Übersicht über das Beispielnetz für das ODEnet

der Implementierung mit Tensorflow in die vorhandenen Differentialgleichungslöser zu groß ist.

Für das Training des ODEnet werden die gleichen Daten wie bei SINDYc verwendet. Wie beim SINDYc-Ansatz, werden die Daten vor dem Netztraining mit einem Skalierer standardisiert. Der verwendete Skalierer ist hierbei sowohl bei den Daten, als auch bei der Zeit, jeweils der gleiche wie bei SINDYc. Dadurch ist eine Vergleichbarkeit zwischen dem SINDYc- und dem ODEnet-Ansatz möglich.

Die in dieser Arbeit verwendeten KNN sind ausschließlich tiefe KNN, da sich bei ersten Tests gezeigt hat, dass mehrere Schichten ein besseres Ergebnis liefern, als nur eine Eingangsschicht, eine verborgene Schicht und eine Ausgangsschicht. Des Weiteren müssen, um das Systemverhalten zu finden und damit das Netz nicht nur eine Trajektorie lernt (Überanpassung), die Startwerte für das Training variiert werden. Deswegen besteht ein Datensatz, den das KNN lernt, aus Trajektorien mit verschiedenen Anfangswerten. Darüber hinaus werden auch Trainingsdatensätze mit verschiedenen Eingängen erstellt, damit das Netz nicht nur das Verhalten des Eingangs lernt, sondern die Systemfunktion. Das Beispielnetz soll Beispiel 1 abbilden. Dafür wird ein KNN mit einer Eingangsschicht mit zwei (bzw., falls der Eingang u vorhanden ist drei) Eingängen erstellt. Darauf folgen $h = 5$ eng verknüpfte Schichten mit jeweils 50 Neuronen und eine Ausgangsschicht mit zwei Neuronen, da es in Beispiel 1 zwei Zustände gibt. Die Aktivierungsfunktionen sind immer ReLU, bis auf die Eingangs- und Ausgangsschicht, in der lineare Aktivierungsfunktionen benutzt werden. Demzufolge hat das Beispielnetz sieben Schichten und 255 (254 ohne Eingang u) Neuronen. Die Zusammensetzung des Netzes ist in Tabelle 1 kompakt dargestellt.

Das Training wird als sogenanntes *Chargentraining* durchgeführt. Diese Trainingsform ist eine Art des unüberwachten Lernens. Um dieser Art des Trainings zu verstehen, bedarf es der Erklärung der Begriffe *Stichprobe*, *Chargengröße* und *Epoche* und deren Zusammenhänge.

Eine Stichprobe ist ein Teil eines Datensatzes. Eine Stichprobe besteht in dieser Arbeit aus zwei aufeinanderfolgenden Datenpunkten. Generell ist es aber auch möglich, dass

ein Datensatz aus nur einer Stichprobe besteht.

Die Chargengröße gibt an, wie viele Stichproben im KNN verarbeitet werden, bevor die Netzparameter verändert werden. Bei dem hier angewendeten Training ist die Chargengröße immer so groß wie die Anzahl der Stichproben. Das bedeutet, dass immer alle Stichproben einmal im KNN verarbeitet werden, bevor es zu einer Veränderung der Netzparameter kommt. Damit ist eine Charge mit einer Epoche gleichzusetzen. Während einer Epoche hat das KNN jede Stichprobe einmal verarbeitet. Die Anzahl der Epochen gibt in dem hier verwendeten Chargentraining an, wie oft jede Stichprobe im KNN verarbeitet wurde. Falls die Chargengröße nicht der Anzahl der Stichproben entspricht, spricht man von einem Mini-Chargentraining, bzw. falls die Chargengröße eins ist, von stochastischem Chargentraining.

Die Epochenanzahl beträgt in dieser Arbeit standardmäßig 2000. Nachdem eine Charge trainiert und prädiziert wurde, werden die Netzparameter θ durch den Optimierer verändert. Diese Art von Training wird dann für mehrere Epochen wiederholt, bis sich ein möglichst kleiner Fehler einstellt.

3.3 Auswertung der Ergebnisse der Testsysteme

Alle Systemidentifikationsansätze wurden mit den oben beschriebenen Systemen getestet. Um sich einen ersten Überblick zu verschaffen, werden zuerst die Ergebnisse gezeigt. Die Diskussion zur Abhängigkeit von verschiedenen Parametern erfolgt in den Abschnitten 3.3.3 und 3.3.4.

Von Interesse ist die Abweichung der prädizierten Daten von den Referenzdaten. Für die zeitliche Betrachtung der Abweichung wird die quadratische Abweichung (QA)

$$\epsilon = ||\boldsymbol{x}_i - \boldsymbol{x}_{i,\mathrm{ref}}||_2^2 \tag{3.10}$$

verwendet.

Zusätzlich erfolgt die Beurteilung der Ergebnisse auch auf Basis eines Gütekriteriums, das oftmals im Zusammenhang mit der Regressionsrechnung verwendet wird; die MQA:

$$\mathrm{MQA} = \frac{1}{n}\sum_{i=1}^{n} |\boldsymbol{x} - \boldsymbol{x}_{\mathrm{ref}}|^2, \tag{3.11}$$

mit der n Zuständen, dem Referenzzustand $\boldsymbol{x}_{ref}$ und dem durch die Identifikationsansätze geschätzten Zustand $\boldsymbol{x}$[1]. Die MQA gibt an, wie sehr ein Punktschätzer um einen zu schätzenden Wert streut [2].

[1] Im folgenden wird für die durch SINDYc geschätzten Zustände $\boldsymbol{x}_{\mathrm{SINDYc}}$ und für die durch ODEnet geschätzten Zustände $\boldsymbol{x}_{\mathrm{ODEnet}}$ verwendet

3.3.1 Ergebnisse des SINDYc-Ansatzes

Für die Tests mit den Beispielsystemen werden die Parameter von SINDYc auf $\lambda = 1, \alpha = 2, D = 10^2$ festgelegt.

Beispiel 1 Für das Beispiel 1 ohne Eingang, wurden die Verläufe in Abbildung 7 aufgezeichnet. Der Grafik ist zu entnehmen, dass die Annäherung durch SINDY hervorragend erfolgt. Die Abweichung im Verlauf der Zeit ist in Abbildung 8 dargestellt. Auch anhand der QA ist zu erkennen, dass SINDY die Koeffizienten sehr gut schätzt.

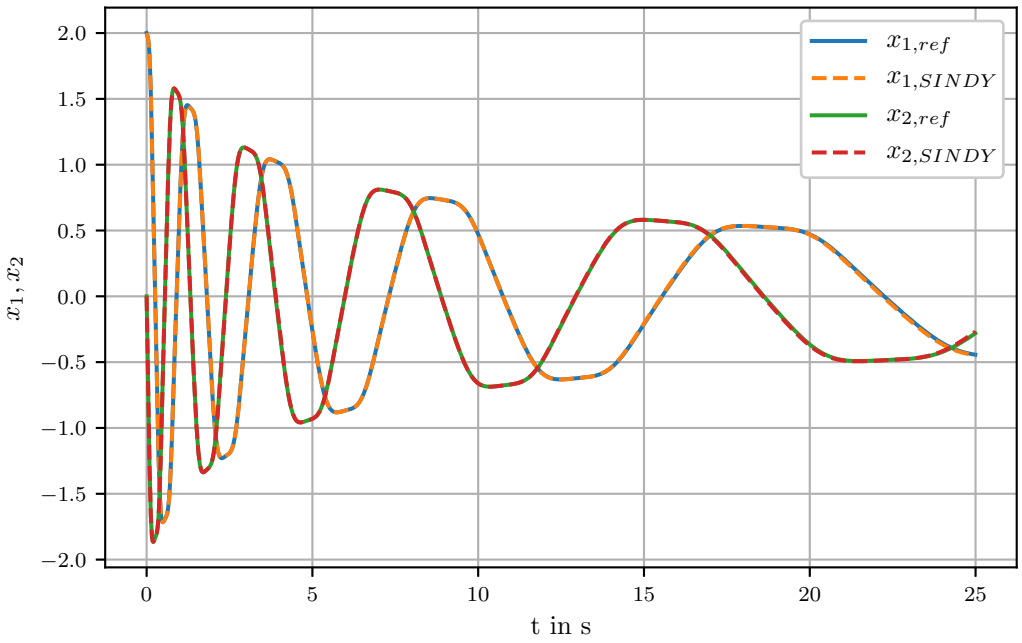

Abbildung 7 – Zustandsverläufe von Beispiel 1 ohne Eingang

Für das Beispiel 1 sind die von SINDY geschätzten Koeffizienten in der folgenden Matrix

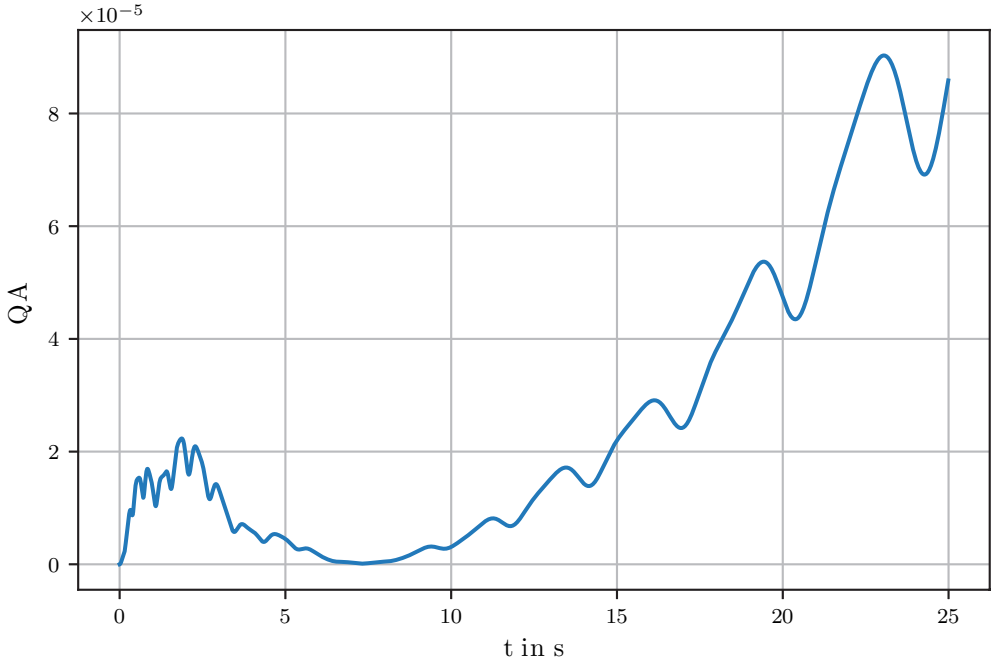

Abbildung 8 – QA von Beispiel 1 ohne Eingang

angegeben. Dabei wurden die oben genannten Parameter und kein Eingang verwendet.

$$\boldsymbol{\Xi} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ -0.10011 & -1.99823 \\ 0 & 0 \\ 0 & 0 \\ 1.99857 & -0.09923 \end{pmatrix}$$

Es ist zu erkennen, dass die Matrix dünn besetzt ist und die Koeffizienten erfolgreich ermittelt wurden. Allerdings gibt es geringe Abweichungen von den eigentlichen Koeffizienten. Dieser Umstand führt zu zunehmenden Abweichungen bei langer Simulationszeit. Die Parameter wurden durch die in Abschnitt 3.2.1 beschriebene Löschung von Elementen in den Matrizen $\boldsymbol{\Theta}$ und $\boldsymbol{\Xi}$ gefunden.

In Abbildung 9 ist Beispiel 1 zusätzlich mit dem Eingang $u = t$ dargestellt. Erneut ist zu sehen, dass SINDYc die Koeffizienten sehr gut findet und somit das System sehr gut annähert, was auch in der in der, in Abbildung 10, dargestellten QA zu erkennen ist. Durch den Eingang $u = t$ kommt es, im Gegensatz zur QA in Abbildung 8, dazu, dass

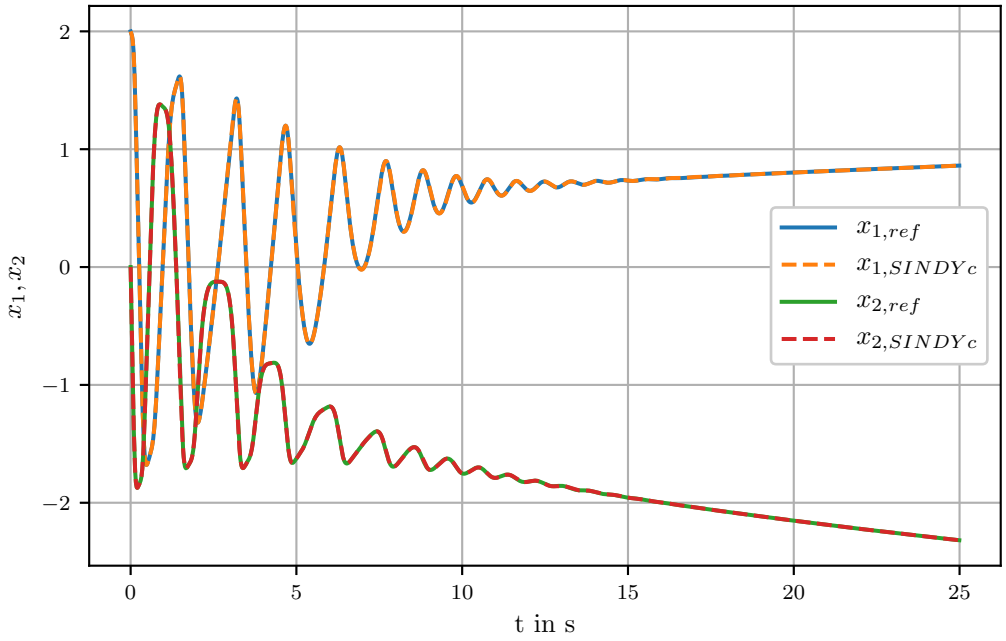

Abbildung 9 – Zustandsverläufe von Beispiel 1 mit Eingang $u = t$

die QA gegen 0 geht.

Beispiel 2 Ähnlich gute Ergebnisse bestätigen sich mit Beispiel 2. Es wird, wie auch in der Literatur, der Eingang $u = 2 \sin t + 2 \sin \frac{t}{10}$ gewählt [5]. Die Abweichungen, die sich in 13 zeigen, sind so gering, dass diese hauptsächlich auf Rundungsfehler und eine zu groß gewählte Abtastrate zurückgeführt werden können.

Beispiel 3 Die Ergebnisse mit SINDYc für Beispiel 3 können die sehr guten Ergebnisse aus den beiden vorangegangenen System nicht bestätigen. Wie in den Abbildungen 14, 15 und 16 zu erkennen ist, sind die Koeffizienten so identifiziert, dass es für kleine Zeiten zu geringen Abweichungen kommt. Für große Zeiten sind die Parameter aber zu ungenau und es kommt zu größeren Abweichungen. Das lässt sich insbesondere mit der QA in Abbildung 16 sehr gut erkennen. Der Grund dafür kann das chaotische Verhalten des Attraktors und der Empfindlichkeit gegenüber geringen Parameterabweichungen sein. Ein weiterer Grund für die Abweichung nach einer bestimmten Zeit konnte durch die Veränderung der Abtastzeit beobachtet werden. Wenn diese sehr klein gewählt wird, kann das System besser identifiziert werden. Auf diese Abhängigkeiten soll in Abschnitt 3.3.3 genauer eingegangen werden.

Abschließend kann zur Identifizierung der Testsysteme mit dem SINDYc-Algorithmus

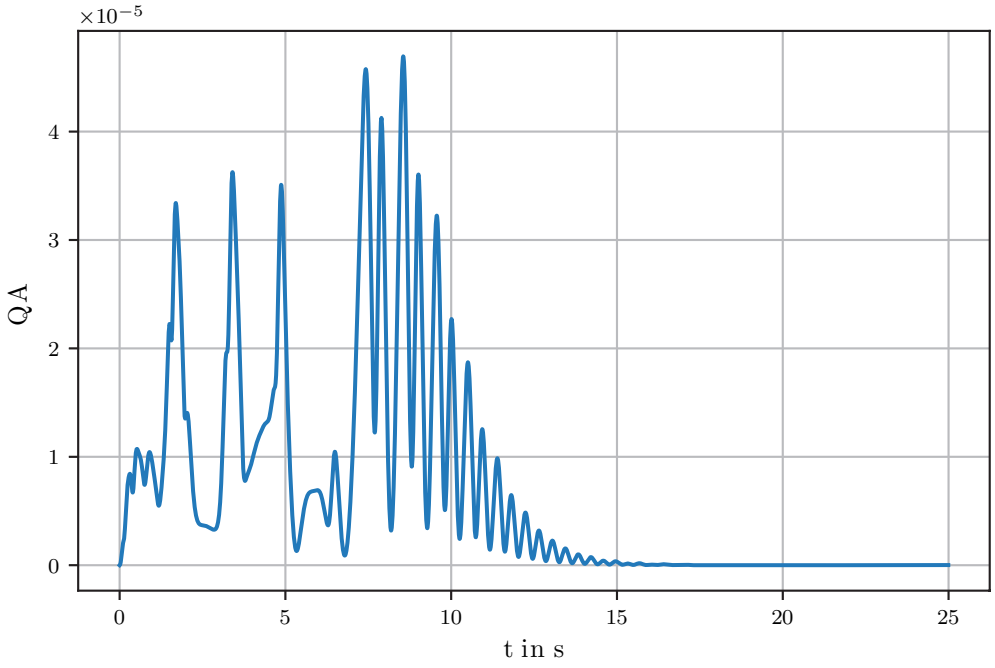

Abbildung 10 – QA von Beispiel 1 mit Eingang $u = t$

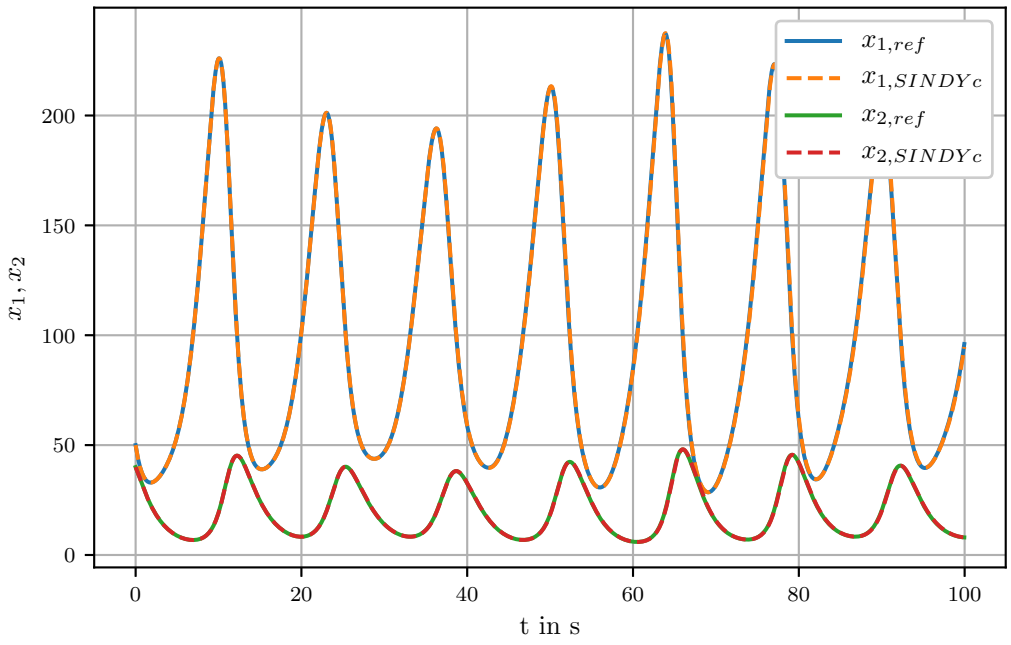

Abbildung 11 – Zustandsverläufe von Beispiel 2 mit dem Eingang $u = 2\sin t + 2\sin \frac{t}{10}$

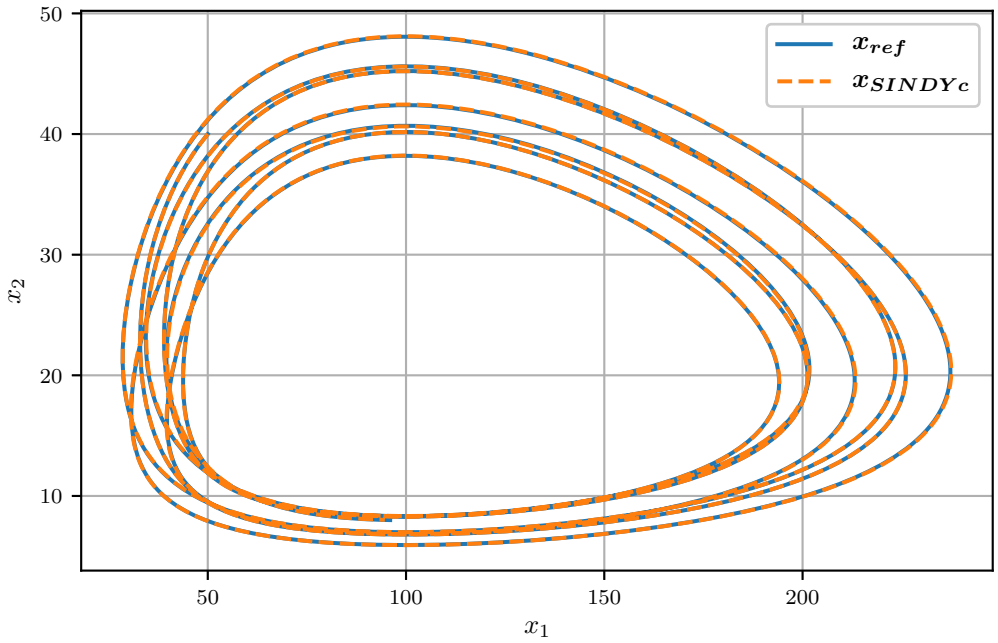

Abbildung 12 – Phasenportrait von Beispiel 2 mit dem Eingang $u = 2\sin t + 2\sin \frac{t}{10}$

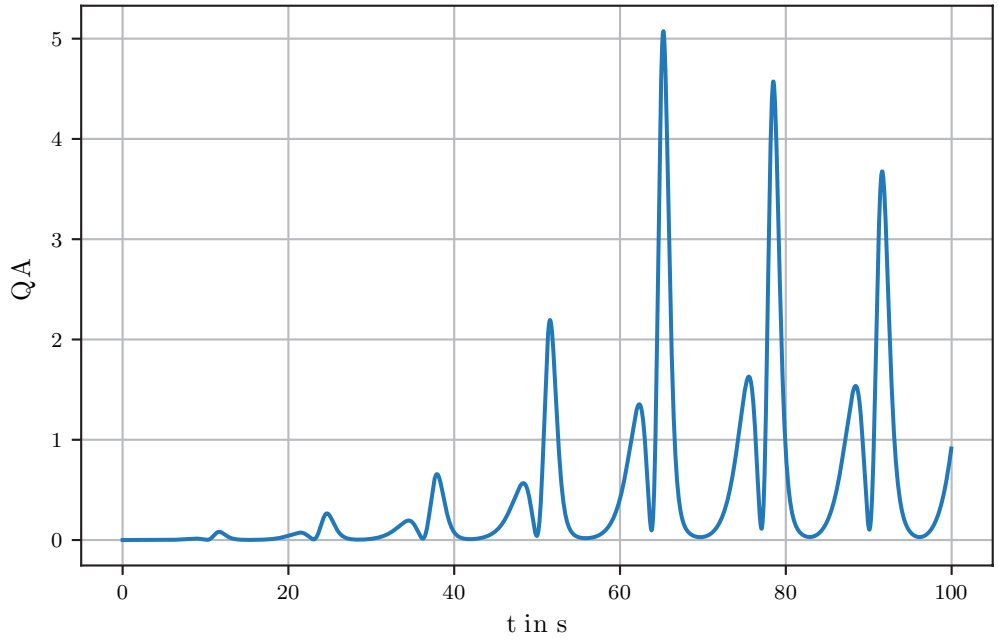

Abbildung 13 – QA von Beispiel 2 mit dem Eingang $u = 2\sin t + 2\sin \frac{t}{10}$

zusammengefasst werden, dass dieser die Erwartungen erfüllt und sehr gute Identifizierungsergebnisse liefert, so lange die Systeme nicht zu komplex werden und die Abtastzeit klein genug gewählt wird. Wie klein die Abtastzeit gewählt werden muss, wird in Abschnitt 3.3.3 diskutiert. Inwiefern dieser Ansatz auch mit dem realen Daten zu zufriedenstellenden Identifizierungen führt, muss geprüft werden.

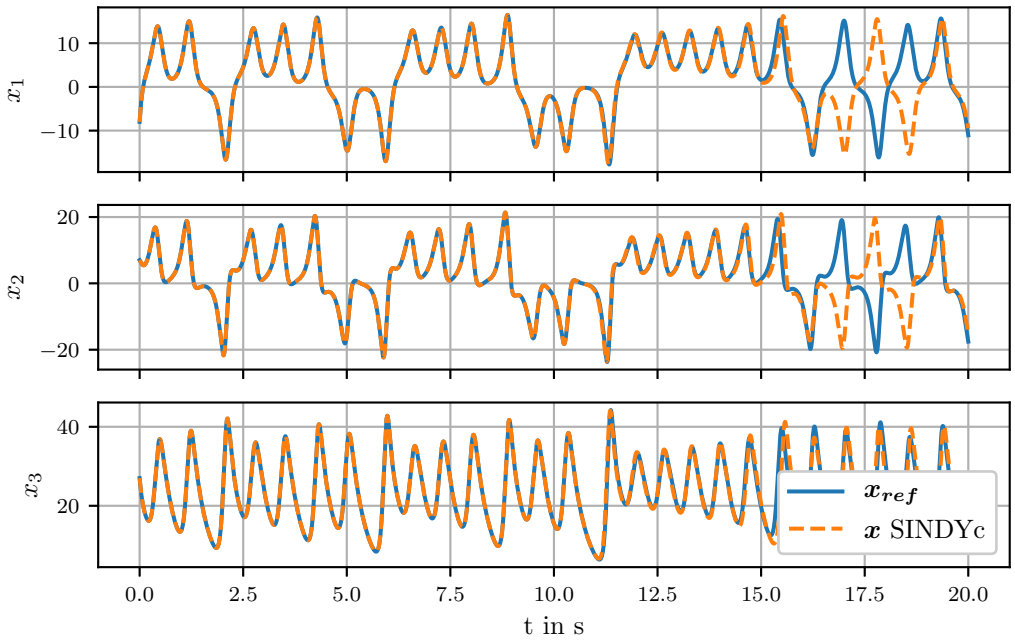

Abbildung 14 – Zustandsverläufe von Beispiel 3 mit dem Eingang $u = 2\sin t + 2\sin\frac{t}{10}$

Einfluss von Rauschen auf SINDYc

Interessant für die reale Anwendung ist der Einfluss von Rauschen auf die Systemidentifikationsansätze. Demzufolge wurden die Trainingsdaten mit mittelwertfreiem weißen Rauschen mit einer Standardabweichung s_r überlagert. Es ist anzunehmen, dass die Identifikation der Systeme gestört wird oder überhaupt nicht mehr möglich ist. Sollte dies der Fall sein, müssen die Daten gefiltert werden, damit diese von SINDYc verarbeitet werden können.

In den Abbildungen 17 und 18 sind die Auswirkungen des Rauschens zu erkennen. Es ist ersichtlich, dass die Systemidentifikation nicht die gleiche Güte hat, wie es ohne Rauschen der Fall ist. Bei $s_r = 0{,}1$ schafft es SINDYc noch gut, dass System zu identifizieren, was bei $s_r = 0{,}3$ insbesondere am Ende nicht mehr behauptet werden kann. Demzufolge müssten die Daten bei $s_r = 0{,}3$ gefiltert werden.

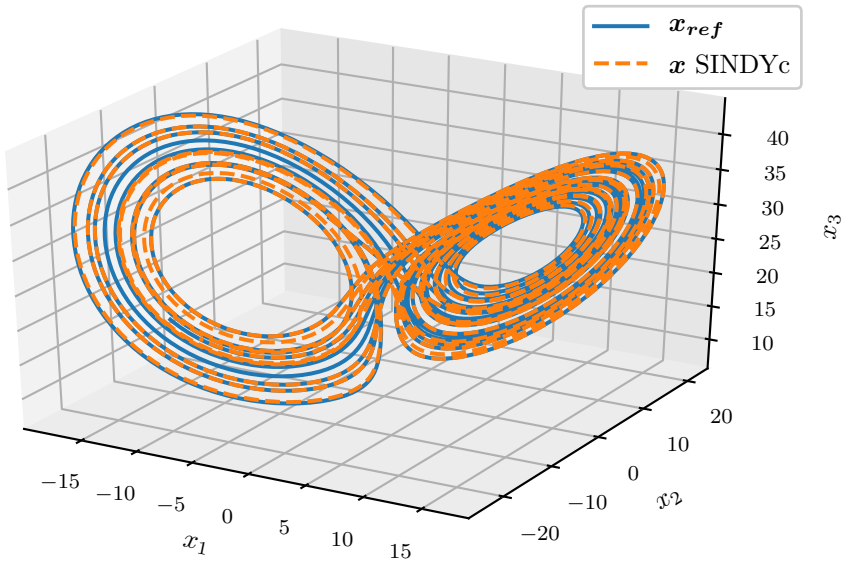

Abbildung 15 – Phasenportrait von Beispiel 3 mit dem Eingang $u = 2\sin t + 2\sin\frac{t}{10}$

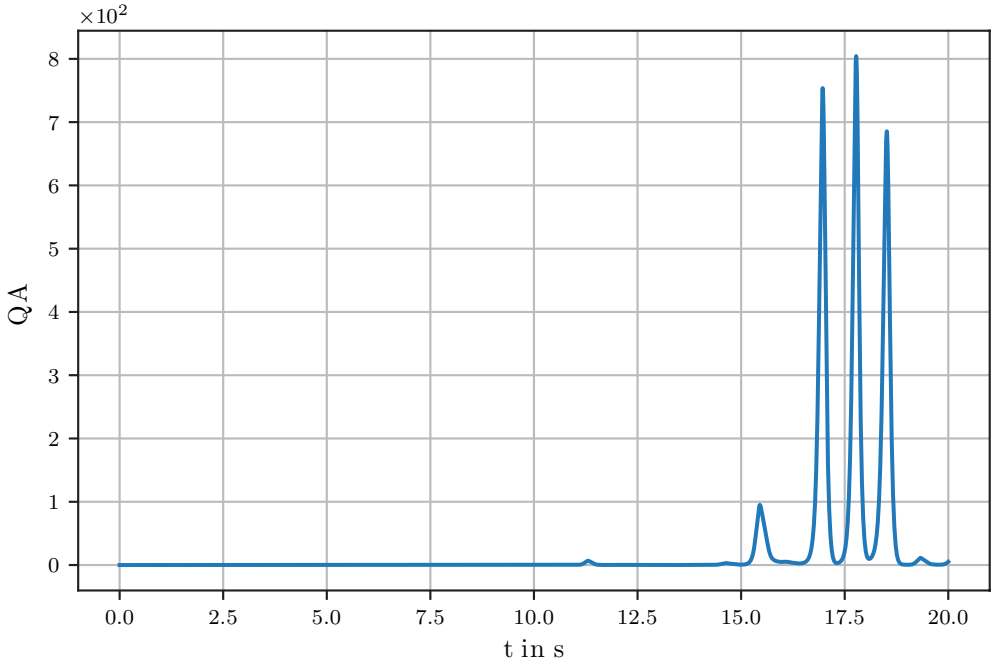

Abbildung 16 – QA von Beispiel 3 mit dem Eingang $u = 2\sin t + 2\sin\frac{t}{10}$

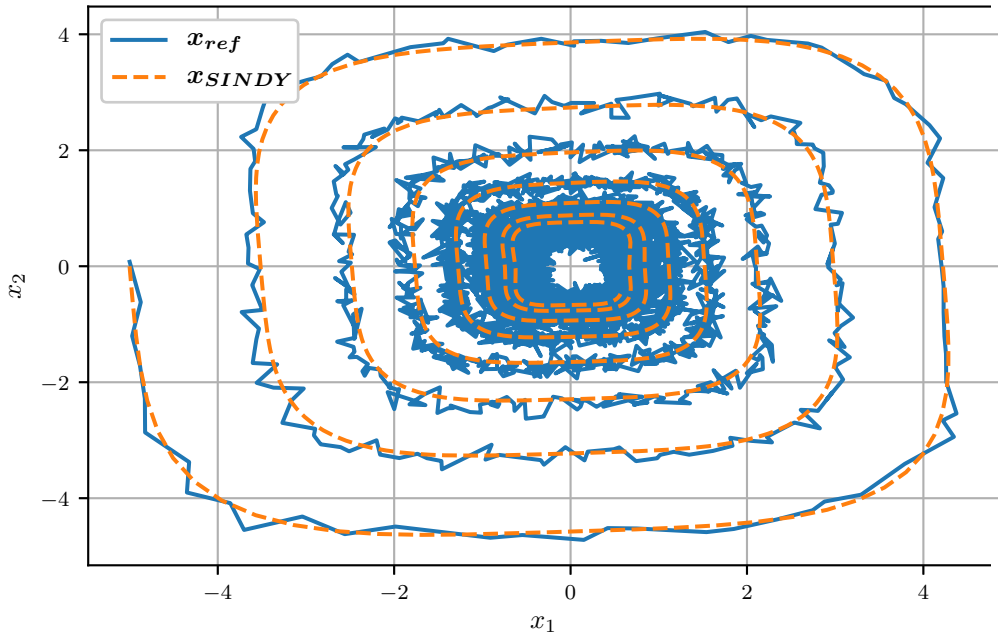

Abbildung 17 – Phasenportrait von Beispiel 1 ohne Eingang mit mittelwertfreiem weißen Rauschen mit $s_r = 0{,}1$ bei SINDYc

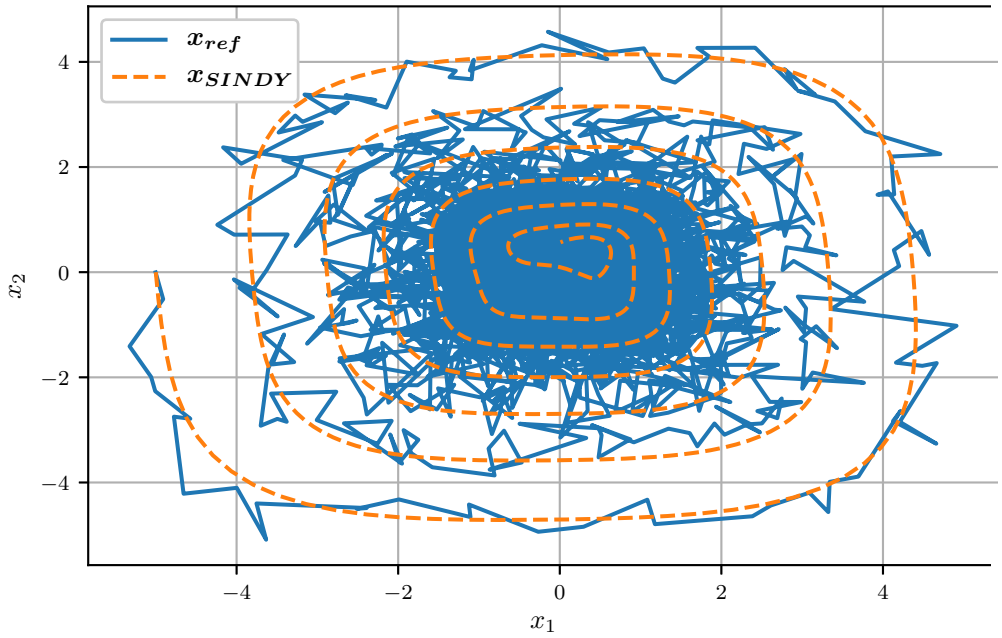

Abbildung 18 – Phasenportrait von Beispiel 1 ohne Eingang mit mittelwertfreiem weißen Rauschen mit $s_r = 0{,}3$ bei SINDYc

3.3.2 Ergebnisse des ODEnet-Ansatzes

Um den ODEnet-Ansatz zu testen, werden die oben eingeführten Beispiele verwendet. Allerdings müssen diese etwas abgewandelt werden, weil sonst die Sinnhaftigkeit eines KNN bzgl. der Generalisierung nicht gegeben wäre.

Entscheidend für belastbare Aussagen über den Erfolg oder Misserfolg des ODEnet-Ansatzes ist die Zusammenstellung der Trainingsdaten. Dabei gilt es eine Überanpassung zu vermeiden und trotzdem eine ausreichend gute Systemidentifikation zu erreichen.

Dafür werden Trainings- und Testdaten für die Beispiele 1 und 2 erstellt. Bei Beispiel 1 wird kein Eingang verwendet, um zu testen, ob der Ansatz überhaupt im Stande ist, nichtlineare Systeme zu identifizieren (Versuch 1). Bei Beispiel 2 werden zwei verschiedene Eingänge verwendet. Einer entspricht dem oben genannten Sinus-Eingang von Beispiel 2, der andere enthält eine Abwandlung im zweiten Sinus-Term (Versuch 2). Außerdem wird bei Beispiel 2 auch ein Datenset mit dem Eingang $u = t$ und dem Sinus-Eingang versucht zu trainieren (Versuch 3). Des Weiteren wird bei den drei Versuchen der Anfangswert über ein bestimmtes Intervall variiert. Demzufolge ergibt sich ein diversifiziertes Datenset, mit dem eine Überanpassung verhindert werden soll.

Zur Verifizierung des Trainingserfolges wird zum einen die MQA während des Trainings, als auch die MQA eines Testdatensatzes verwendet, denn so lässt sich eine Überanpassung erkennen.

Versuch 1 Das Ergebnis des Trainings für Beispiel 1 ist ihn Abbildung 19 als Phasenportrait dargestellt. Es ist zu erkennen, dass das Ergebnis nicht optimal ist. Das ODEnet schafft es nicht, die Systemgleichungen genau zu ermitteln. Zwar schafft es der Ansatz, das System ungefähr zu erlernen, jedoch ist die Abweichung, insbesondere in der Mitte (also am Ende des zeitlichen Verlaufs), groß. Dieser Umstand kann auf das Kostenfunktional zurückgeführt werden, welches die MQA ist. Da die Werte in der Mitte kleiner sind als diejenigen an den Rändern, versucht der Optimierer hauptsächlich die relativ großen Abweichungen an den Rändern zu optimieren. Im zeitlichen Verlauf führt dies zu einem guten Ergebnis am Anfang, aber einem schlechten am Ende. Dies ist in Abbildung 20 dargestellt. Mit einem zeitlich gewichteten Kostenfunktional könnte dem entgegengewirkt werden.

Des Weiteren fällt auf, dass die Anzahl der Datenpunkte keinen entscheidenden Einfluss auf das Ergebnis hat, denn das ODEnet schafft es, das System so genau zu lernen, dass die Trajektorie zwischen den einzelnen Abtastzeitpunkten ausreichend genau ist.

Die MQA über die trainierten Epochen ist in Abbildung 21 dargestellt. Anhand dieser ist festzustellen, dass das Ergebnis mit einer größeren Anzahl trainierter Epochen besser wird. Dabei erfolgt am Anfang eine schnelle Optimierung der Parameter, später schwächt sich diese deutlich ab und stagniert sogar. Dies könnte ein Anzeichen für ein ausreichend

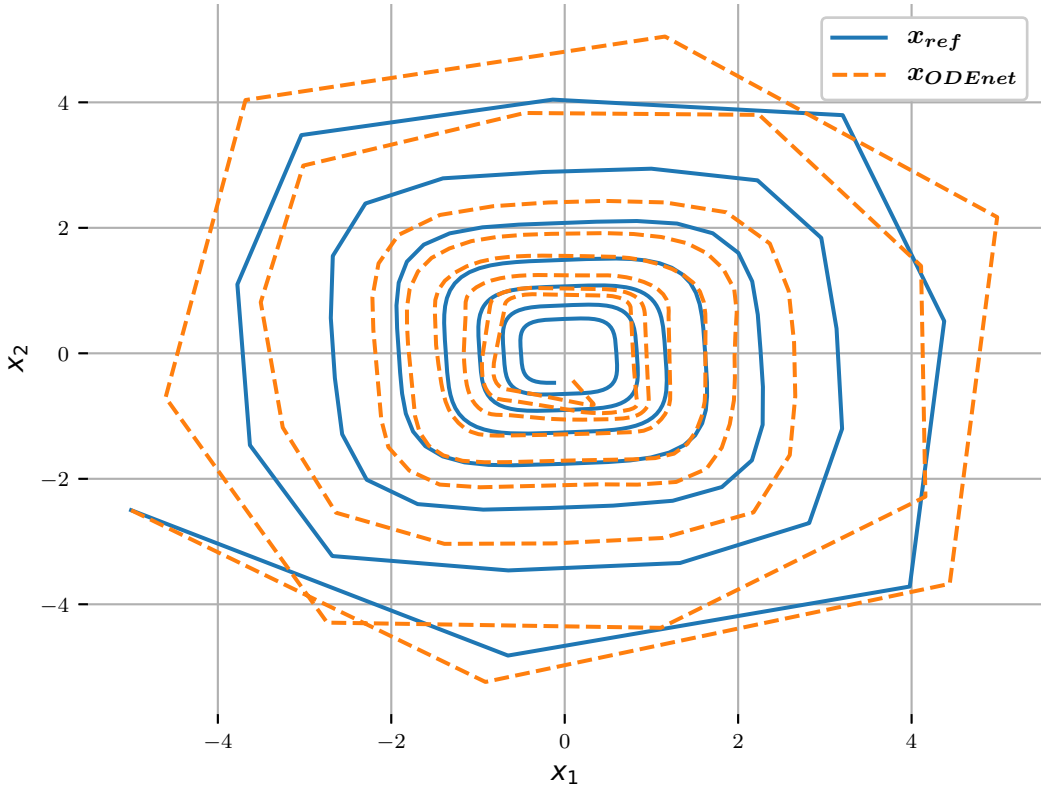

Abbildung 19 – Phasenportrait für ODEnet für das Beispiel 1 ohne Eingang und Anfangswertintervall $x_0 = [-5, 5]$ und zehn Datensets

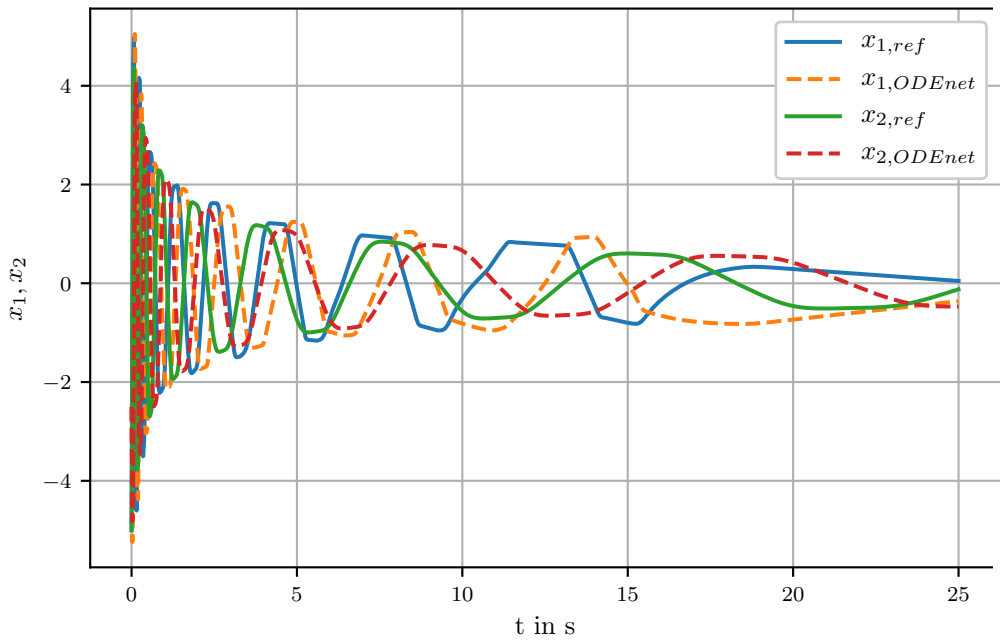

Abbildung 20 – Zustandsverläufe für ODEnet für das Beispiel 1 ohne Eingang und Anfangswertintervall $x_0 = [-5, 5]$ und zehn Datensets

gutes Trainingsergebnis sein.

Anhand des Testdatensatzes in Abbildung 21 zeigt sich auch, dass es nach ca. 150 Epochen zu einer Überanpassung kommt, da die MQA wieder steigt. Das Training sollte an dieser Stelle also abgebrochen werden, weil das KNN die Daten sonst auswendig lernt und nicht mehr generalisiert. Es ergibt sich zwar nach weiteren Epochen eine Verringerung der MQA. Dies ist jedoch eher zufällig und kann nicht genutzt werden.

Versuch 2 In Abbildung 22 ist das Ergebnis von Beispiel 2 mit den zwei unterschiedlichen Sinus-Eingängen zu erkennen. Erkennbar ist, wie bereits im vorhergehenden Versuch, dass der ODEnet-Ansatz es zwar schafft, das Systemverhalten ungefähr zu erlernen, die genaue Identifikation schlägt jedoch fehl. Die Abweichung hat sich im Gegensatz zum Beispiel 1 etwas erhöht, was auf die Verwendung von Trainingsdaten mit zwei unterschiedlichen Eingängen zurückzuführen ist. Auch wenn sich die Eingänge sehr ähneln, schafft es das ODEnet nicht, diese zu Erlernen. Ein Grund hierfür könnte die Verwendung einer zu geringen Menge von unterschiedlichen Trainingsdatensätzen sein, jedoch verlängert sich das Training mit Zunahme der Datensätze erheblich.

Auch für dieses Training wurde die MQA für das Training und den Test in Abbildung 23 dargestellt. Es ist erneut zu sehen, dass im Training eine fortlaufende Optimierung

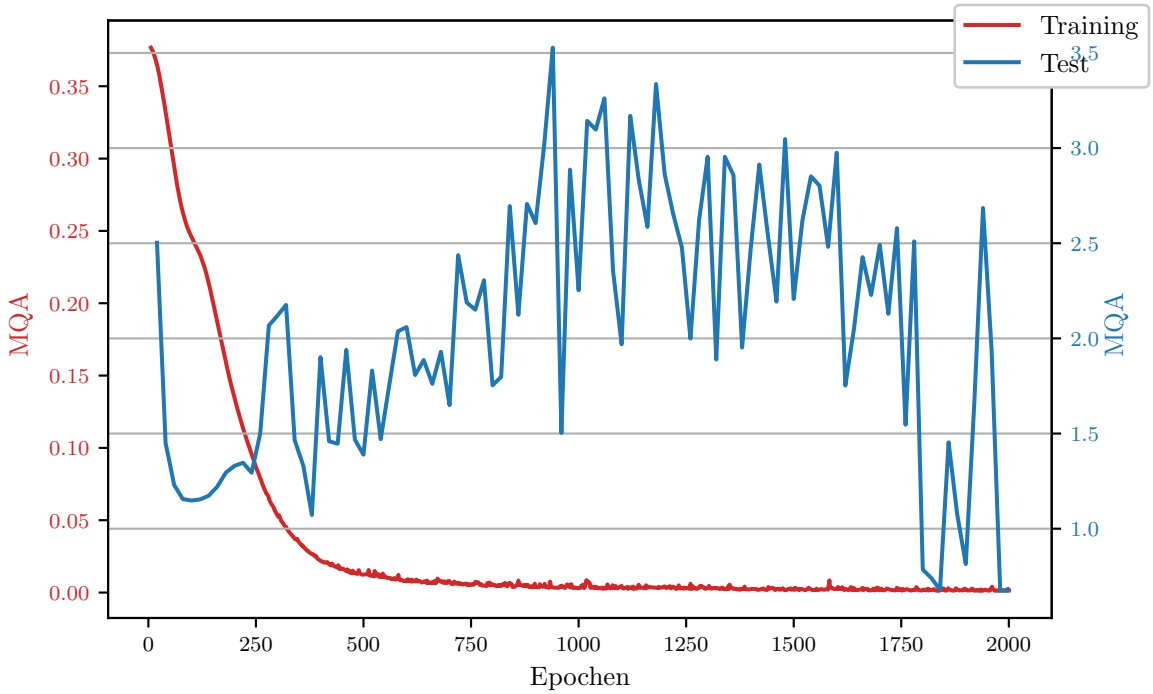

Abbildung 21 – QA des Trainings und des Tests für ODEnet für das Beispiel 1 ohne Eingang und Anfangswertintervall $x_0 = [-5, 5]$ und zehn Datensets

stattfindet. Dass die MQA etwas volatiler als in Abbildung 21 aussieht, liegt an der Verwendung eines anderen Optimierers, der bei diesem Beispiel ein besseres Ergebnis erzielt hat.

Bei einem Vergleich mit dem Testdatenset zeigt sich jedoch, dass sich eine Überanpassung einstellt. Auch hier sollte das Training nach ca. 750 Epochen abgebrochen werden. An dieser Stelle ist die MQA bei ca 1. Es könnte jedoch auch weiter trainiert werden und der Abbruch erst nach 1750 Epochen erfolgen. Dafür ist es notwendig das Ergebnis des KNN an dieser Epoche zu überprüfen.

Versuch 3 Der letzte Versuch basiert auch auf Beispiel 2. Die Zusammensetzung des Datensets gleicht dem vorangegangenen Versuch, mit dem Unterschied, dass als Eingänge der Sinus-Eingang und der lineare Eingang aus Beispiel 1 verwendet wird. Als Test soll auch hier der lineare Eingang genutzt werden. Abbildung 24 stellt das Ergebnis dar. Wie schon in den ersten beiden Versuchen, schafft es das KNN nicht, das System genau zu identifizieren. Erschwerend kommt hier dazu, dass auch der Eingang offensichtlich nicht richtig identifiziert und gelernt wird. Der Sinus-Eingang unterscheidet sich deutlich vom linearen Eingang, weswegen das KNN nicht im Stande ist den Eingang im Testset richtig zu berechnen. Daraus ist zu schließen, dass der ODEnet-Ansatz nur für Systeme geeignet ist, bei denen sich der Eingang des Trainings nicht entscheidend

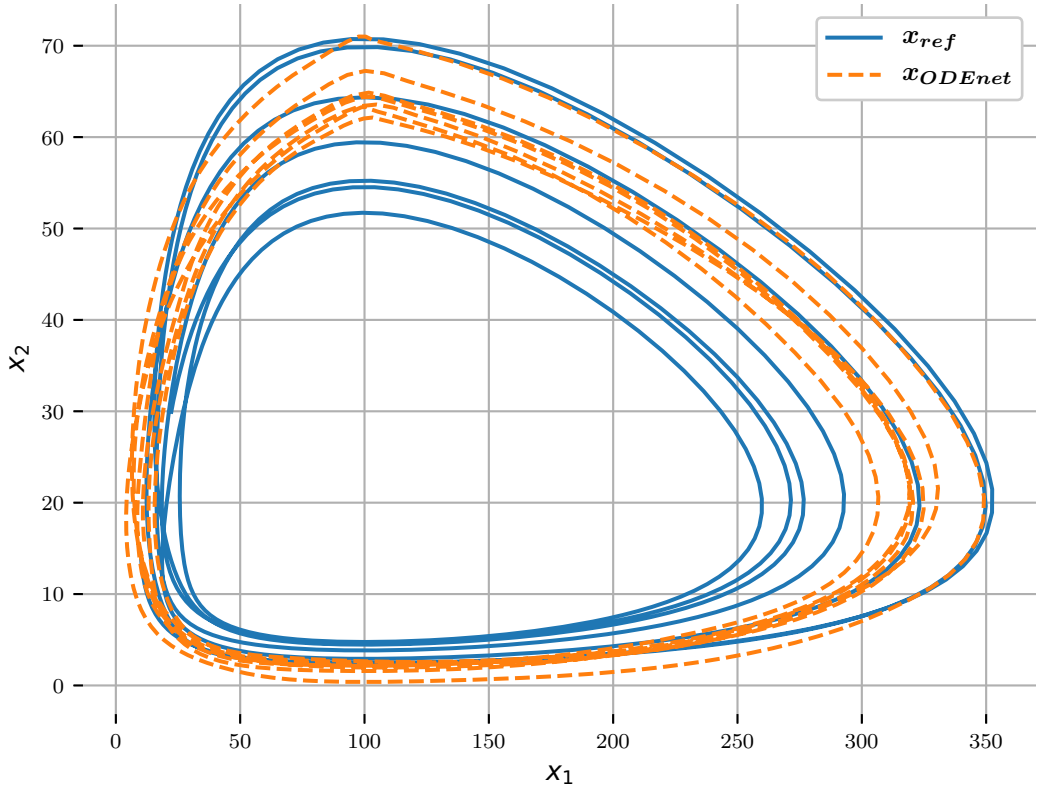

Abbildung 22 – Phasenportrait für ODEnet für das Beispiel 2 mit zwei unterschiedlichen Sinus-Eingängen und dem Anfangswertintervall $x_0 = [10, 50]$ und zehn Datensets

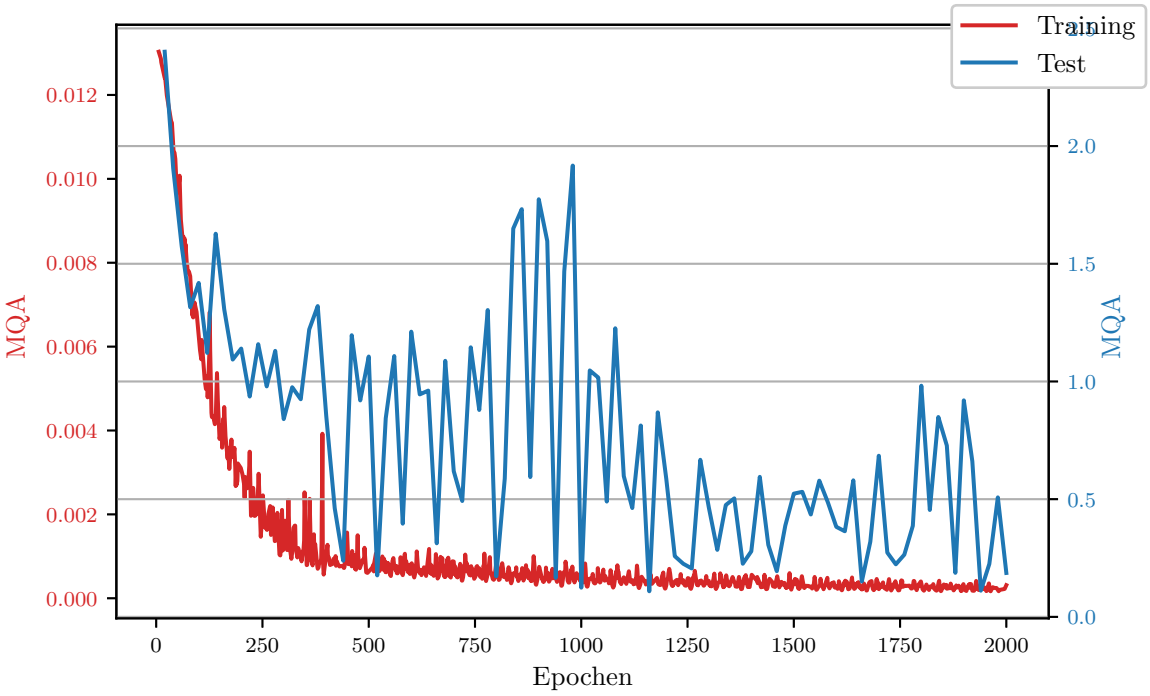

Abbildung 23 – QA des Trainings und des Tests für ODEnet für das Beispiel 2 mit zwei verschiedenen Sinus-Eingängen und einem Anfangswertintervall $x_0 = [10, 50]$ und zehn Datensets

vom Eingang des Tests ändert. Die Variation im Eingang sollte also möglichst klein gehalten werden.

Die Verläufe der MQA ist sehr ähnlich zu den beiden vorangegangenen Versuchen und sollen deswegen hier nicht noch einmal gezeigt werden. Auch in diesem Versuch kommt es nach ca. 400 Epochen zur Überanpassung, weswegen das Training abgebrochen werden sollte.

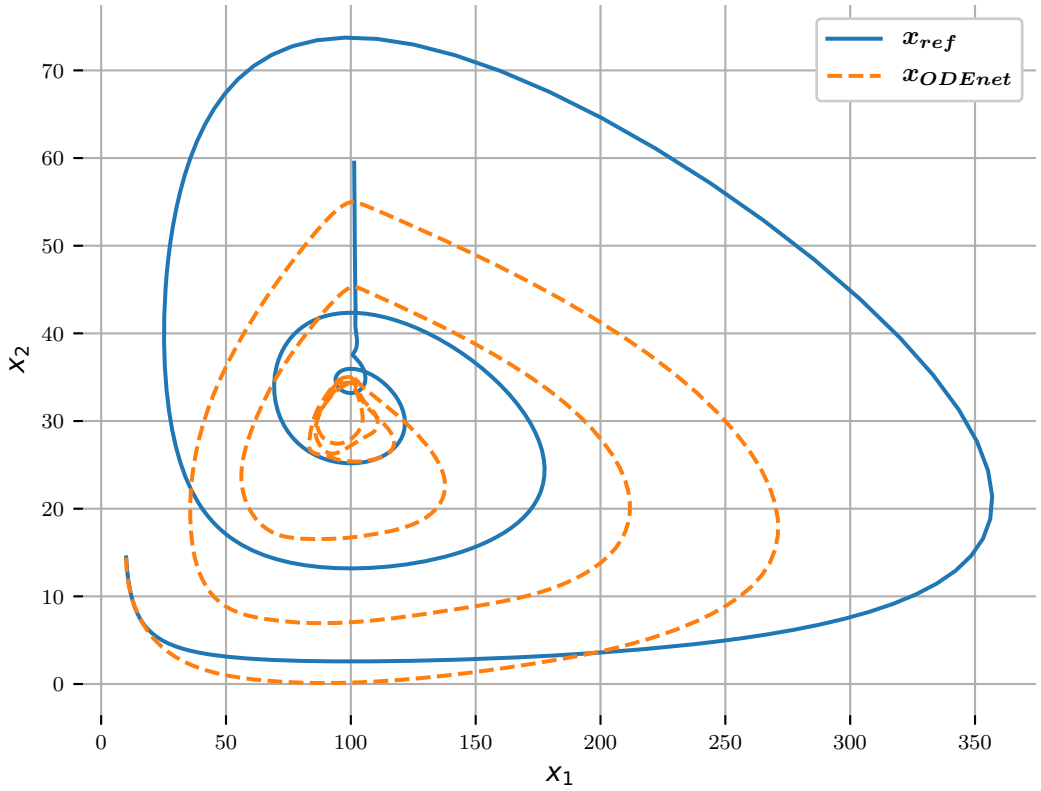

Abbildung 24 – Phasenportrait des Tests für ODEnet für das Beispiel 2 mit zwei verschiedenen Eingängen und einem Anfangswertintervall $x_0 = [10, 50]$ und zehn Datensets

Einfluss von Rauschen auf das ODEnet

Wie auch bei SINDYc wird beim ODEnet Beispiel 1 mit weißem Rauschen mit $s_r = 0{,}1$ überlagert. Interessant ist an dieser Stelle, ob das ODEnet vom Rauschen gestört wird, oder ob es es weitestgehend gut verarbeiten kann. Das Ergebnis des Tests ist in Abbildung 25 zu sehen. Zu erkennen ist, dass der Anfang der Trajektorien noch gut erlernt wird. Sobald das Rauschen die Trajektorien zu stark überlagert, wird das

Ergebnis zunehmend schlechter und für große Zeiten falsch. Demzufolge erübrigt sich auch der Versuch mit $s_r = 0{,}3$.

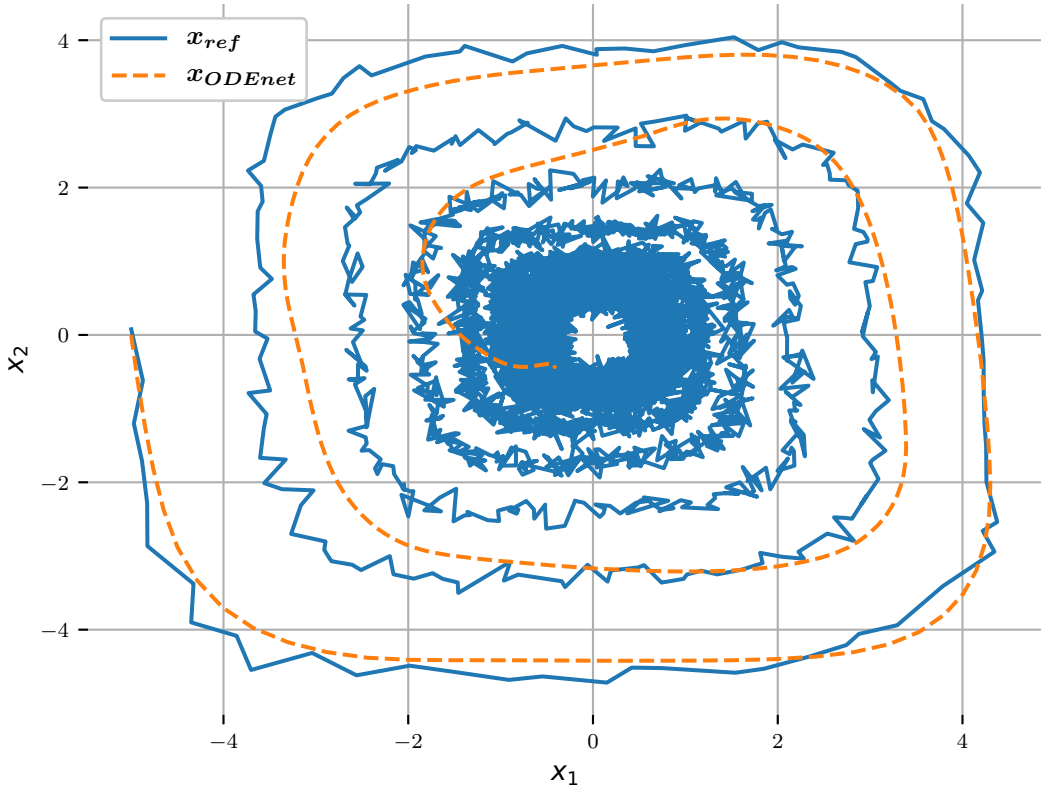

Abbildung 25 – Phasenportrait von Beispiel 1 ohne Eingang mit mittelwertfreiem weißen Rauschen mit $s_r = 0{,}1$ beim ODEnet

Zusätzlicher Test mit geomspace Zusätzlich wurde beim ODEnet Versuch 1 mit Beispiel 1 noch leicht modifiziert. Wie bereits in Abbildung 19 aufgefallen ist, gibt Probleme durch die äquidistant gewählten Abtastzeitpunkte im Phasenportrait. Durch die großen Änderungen am Anfang ist das speziell für dieses System aber nicht vorteilhaft. Deswegen wurde ein neuer Datensatz erstellt, bei dem die Zeitpunkte logarithmisch verteilt sind, in der Hoffnung, dass der Optimierer bessere Parameter findet, um die Punkte am Anfang und ganz am Ende (weil dort dann weniger Punkte sind) besser zu treffen. Das wurde mit der Python-Funktion *geomspace* bewerkstelligt.

Das Ergebnis des Trainings ist in Abbildung 26 dargestellt. Hier schafft es der Algorithmus erstaunlicherweise nicht, das System besser zu identifizieren, als bei der äquidistanten Aufteilung der Datenpunkte. Die Abweichung ist nur am Anfang gering.

Desto mehr das System sich in die Mitte zusammenzieht, desto schlechter wird das Ergebnis. Es könnte sein, dass für die logarithmische Aufteilung zu wenig Punkte am Ende vorhanden sind.

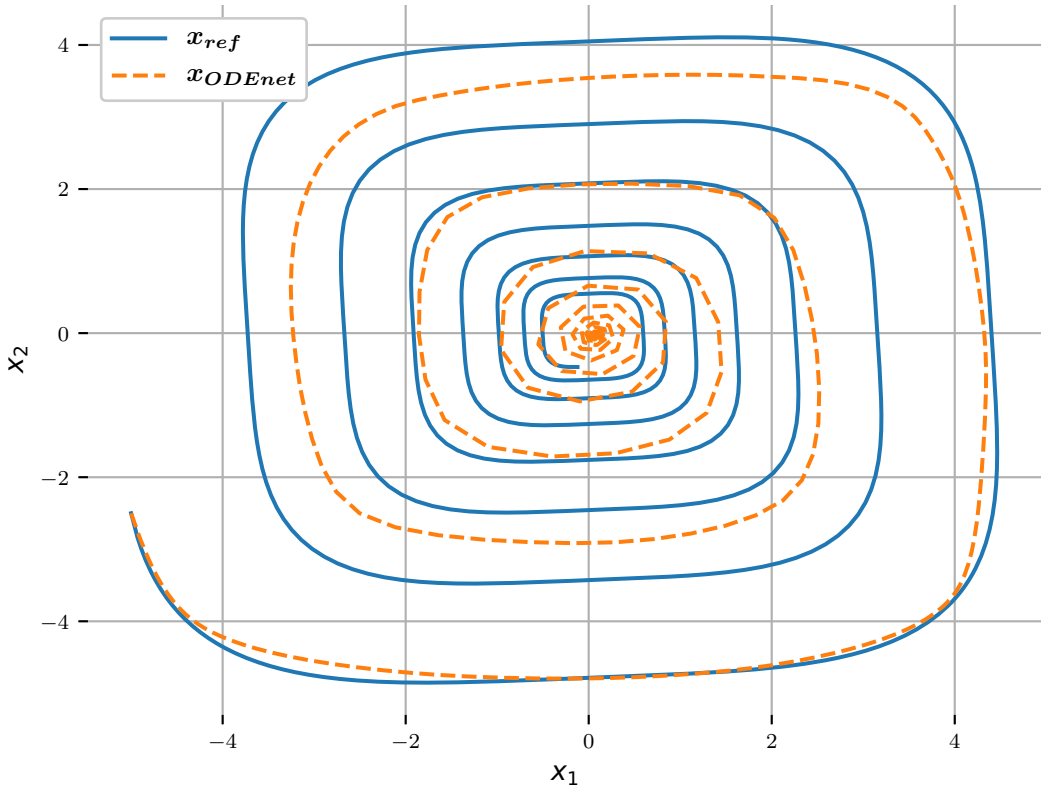

Abbildung 26 – Phasenportrait für ODEnet für das Beispiel 1 ohne Eingang und Anfangswertintervall $x_0 = [-5, 5]$ und zehn Datensets mit logarithmischer Datenaufteilung

3.3.3 Abhängigkeiten von ausgewählten Parametern bei SINDYc

Bei den Versuchen zeigten sich Abhängigkeiten von verschiedenen Parametern. Auf die Abhängigkeit

- von der Abtastzeit,
- von der Simulationszeit,
- vom 1-Norm-Penalisierungsparameter λ,

– vom Potenzfaktor D,

– und vom Polynomgrad α der gewählten Ansatzfunktionen

soll im folgenden weiter eingegangen werden.

Abtastzeit Beim Testen der Systeme war auffällig, dass sehr unterschiedliche Ergebnisse erzielt wurden, sobald die Abtastzeit der realen Daten verändert wird. Generell kann gesagt werden, dass das Ergebnis bei geringer Abtastzeit wesentlich besser ist. Dieser Umstand wird in Abbildung 27 gezeigt. Es ist zu beachten, dass die y-Achse logarithmisch eingeteilt ist. Mit kleiner werdender Abtastzeit verringert sich die QA. Anzunehmen ist, dass der SINDYc-Algorithmus wesentlich mehr Datenpunkte bei eine geringen Abtastzeit zur Verfügung hat und damit das Optimierungsproblem aus (2.21) genauer lösen kann.

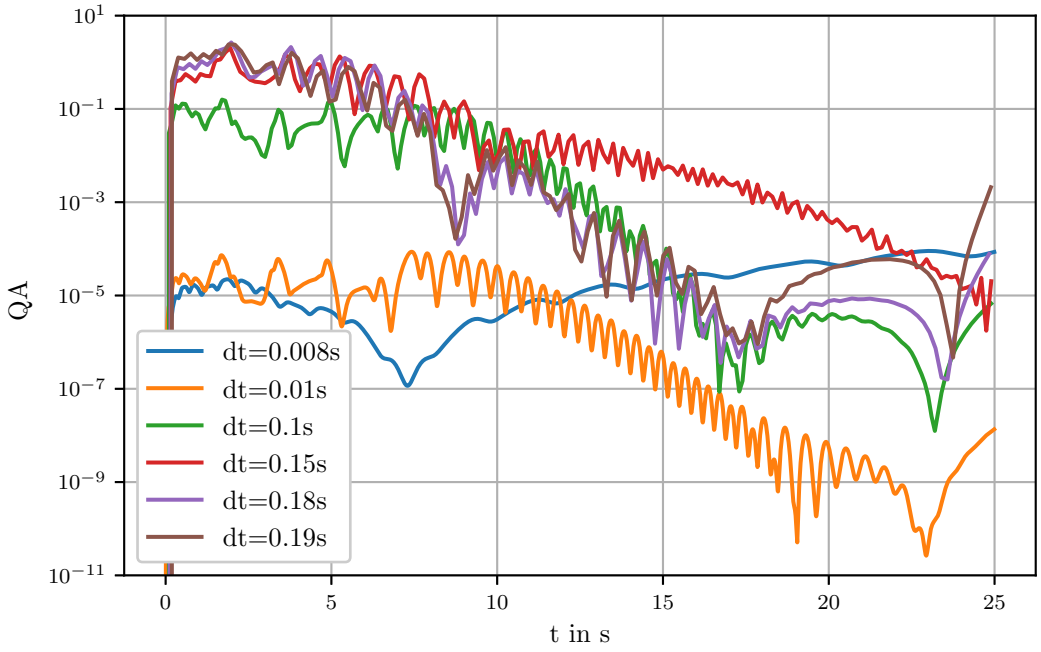

Abbildung 27 – QA bei verschiedenen Abtastzeiten. Es wurde 1 mit den oben genannten Parametern und mit Eingang verwendet.

Simulationszeit Abbildung 27 zeigt, dass sich mit fortschreitender Zeit t das Ergebnis leicht verschlechtert. Dieser Effekt wird deutlicher in Abbildung 28 gezeigt. Die QA ist für kleine Zeiten annähernd gleich, verschlechtert sich dann aber mit voranschreitender Zeit um mehrere Dekaden. Dennoch ist der Fehler in einem sehr kleinen Bereich. Allerdings

könnten Probleme auftreten, wenn es sich um sehr langsame Prozesse handelt, die nicht ausreichend abgetastet werden. Ursache hierfür ist die ungenau Berechnung der Koeffizienten. Auch durch mehrmaliges Ausführen von SINDYc und durch Nutzen von D konnten die Parameter nicht genau gefunden werden.

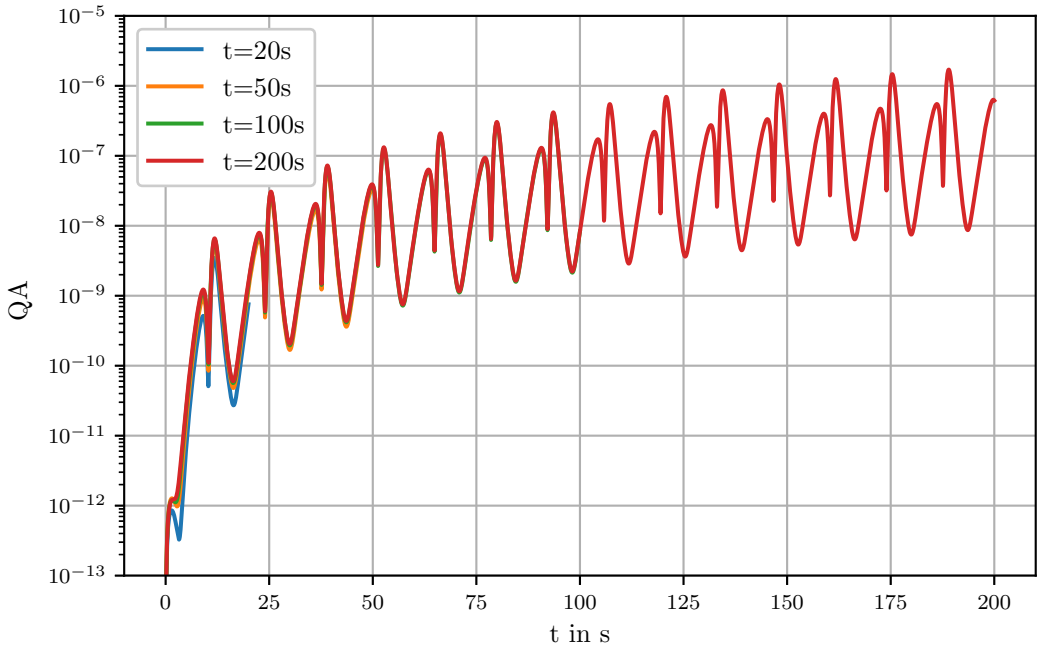

Abbildung 28 – QA bei unterschiedlichen Simulationszeiten. Es wurde das Beispiel 1 mit den oben genannten Parametern und mit Eingang verwendet.

1-Norm-Penalisierungsparameter λ Der Einfluss von λ ist entscheidend für die Genauigkeit der Koeffizienten und die dünne Besetztheit der Matrix $\boldsymbol{\Xi}$. Um diesen Einfluss zu untersuchen, wurde das Beispiel 1 für unterschiedliche λ identifiziert. λ wird im Intervall $[0, 1000]$ variiert und die QA für einige ausgewählte λ aufgezeichnet. Die Darstellung der QA ist Abbildung 29 zu entnehmen. Auffallend hierbei ist, dass die QA über mehrere Dekaden steigt, wenn λ vergrößert wird. Bis zu einem Wert von $\lambda = 10$ scheint nur ein geringer Einfluss zu herrschen. Für größere λ ist die QA ca. $1,5$ Dekaden größer. Durch einen größeren Einfluss von λ werden mehr Koeffizienten verringert. Es zeigt sich bei der Auswertung der Koeffizienten, dass die LASSO-Regression kleine Koeffizienten nicht automatisch auf 0 setzt, sondern nur weiter verringert. Auch scheint der Algorithmus nicht automatisch einen guten Ausgleich für die verkleinerten Koeffizienten bei größeren Koeffizienten zu berechnen. Aufgrund dessen wurde noch die D-Methode implementiert, um die kleinen Koeffizienten auf 0 zu setzen. Der Einfluss der D-Methode wird im nächsten Abschnitt beschrieben. λ sollte für die realen Systeme

zuerst klein gewählt werden, weil es sonst der Fall sein kann, dass wichtige Koeffizienten zu sehr penalisiert werden.

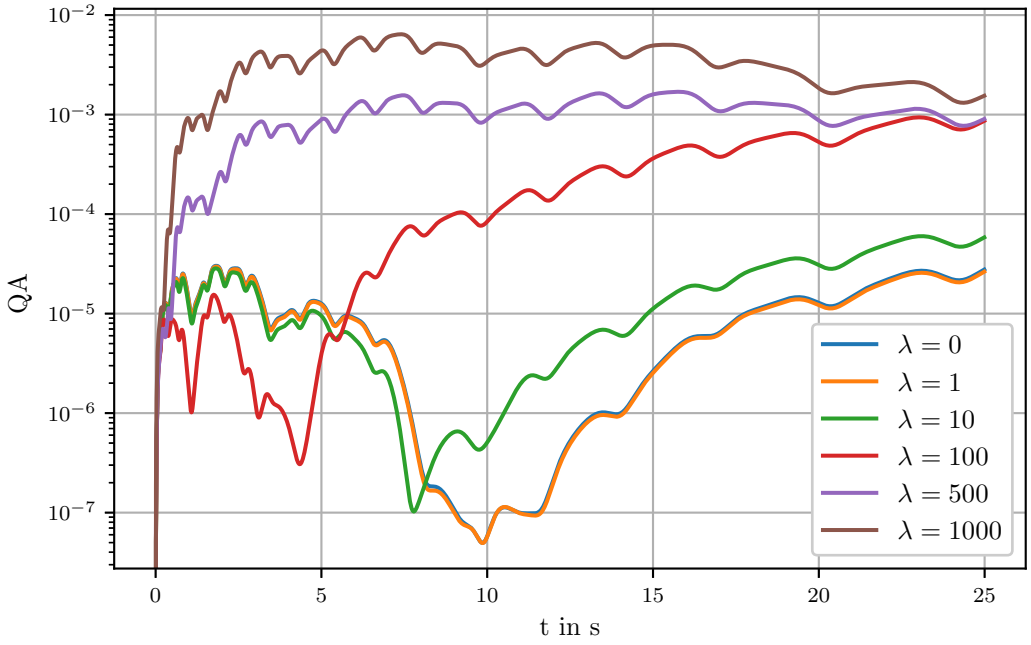

Abbildung 29 – QA unterschiedlicher λ. Es wurde das Beispiel 1 mit den oben genannten Parametern und mit Eingang verwendet.

Potenzfaktor D Wie oben beschrieben, hat λ einen entscheidenden Einfluss auf die Koeffizienten. Um die dünne Besetztheit noch weiter zu erhöhen und wurde D und die Reduzierung der Dimension von $\boldsymbol{\Xi}$ während des Trainings eingeführt. Der Einfluss von D soll hier untersucht werden. Dafür wird D im Intervall von $[10^1, 10^4]$ variiert und die Koeffizienten berechnet. Wie Abbildung 30 zeigt, erhöht sich die QA um ca. eine Dekade. Für $D = 10^3$ und $D = 10^4$ werden fast die gleichen Koeffizienten berechnet. Dadurch liegen die Abweichungen nahe beieinander. Die Vergrößerung der QA für $D = 10^2$ und $D = 10^1$ ist wahrscheinlich auf die Elimination der kleineren Koeffizienten zurückzuführen. Es ist realistisch, dass die LASSO-Regression kleine Koeffizienten berechnet, um Abweichungen vom Referenzsystem auszugleichen. Sobald diese Koeffizienten eliminiert werden, kann der Algorithmus die verbleibenden neuen Koeffizienten nicht ausreichend genau berechnen, um den entstandenen Fehler auszugleichen. Deshalb muss eine Abwägung zwischen dünner Besetztheit der Matrix $\boldsymbol{\Xi}$ und der Genauigkeit der Systemidentifikation getroffen werden. Anfänglich sollte der Einfluss von D gering gehalten werden um das Systemverhalten exakt identifizieren zu können. Sobald die wichtigen Koeffizienten bekannt sind, kann D erhöht und damit geringe Koeffizienten

eliminiert werden. Zur Verdeutlichung der dünnen Besetztheit wird die Matrix $\boldsymbol{\Xi}$ farblich kodiert in Abbildung 31 dargestellt.

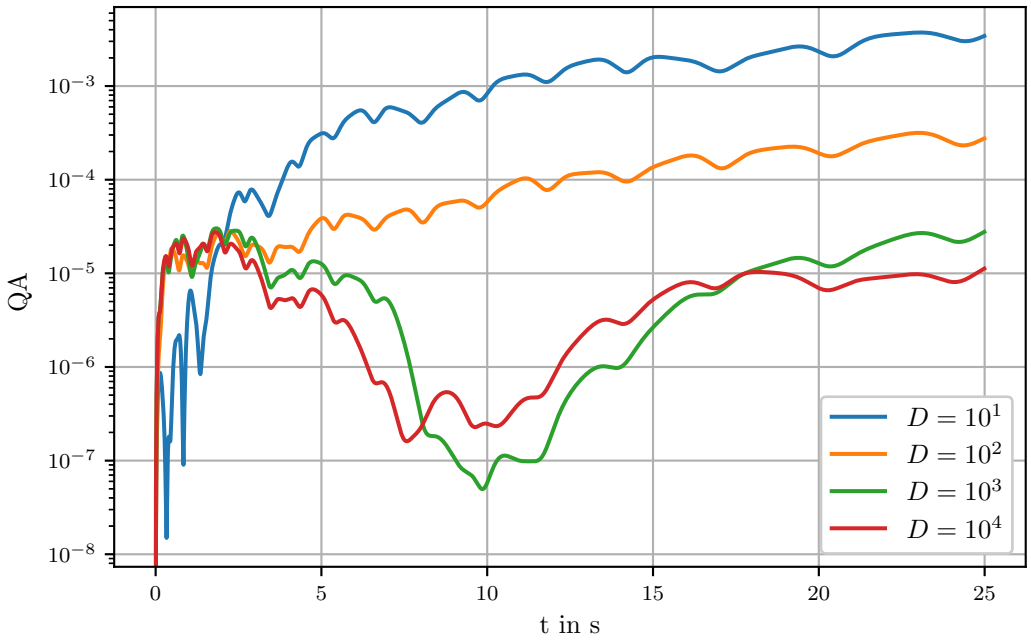

Abbildung 30 – QA unterschiedlicher ΔD. Es wurde das Beispiel 1 mit den oben genannten Parametern und mit Eingang verwendet.

Polynomgrad α der gewählten Ansatzfunktionen Die Ansatzfunktionen haben eine entscheidende Bedeutung beim SINDYc-Ansatz, weil sie mit Koeffizienten belegt werden. Es ist anzunehmen, dass bei geringem Polynomgrad α der Ansatzfunktionen kein gutes Ergebnis erzielt wird. Im Gegensatz dazu ist anzunehmen, dass bei einem zu hohem Grad, der SINDYc-Ansatz diese Ansatzfunktionen nicht berücksichtigt und nur die nötigen Ansatzfunktionen mit Koeffizienten belegt. Dies entspricht exakt dem Sinn und Zweck der LASSO-Regression.

Um diese Annahmen zu testen, wurde Beispiel 1 mit drei unterschiedlichen Graden von Ansatzfunktionen getestet. Das Ergebnis ist in Abbildung 32 dargestellt. Wider erwarten ist das Ergebnis für Ansatzfunktionen vierten Grades schlechter als für Ansatzfunktionen des dritten Grades. Der SINDYc-Ansatz schafft es mit der LASSO-Regression nicht exakt, die richtigen Ansatzfunktionen sofort zu finden. Vielmehr es muss ein Vorwissen über die im System enthaltenen Funktionen vorhanden sein. Dagegen ist das Ergebnis bei Ansatzfunktionen zweiten Grades wie erwartet schlecht. Der Algorithmus hat keine Möglichkeit die Koeffizienten so zu berechnen, dass ein gutes Ergebnis erzielt wird. Ursache hierfür ist, dass die Ansatzfunktionen zweiten Grades nicht zur

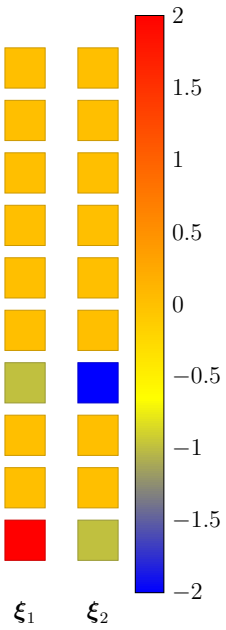

Abbildung 31 – Darstellung der Koeffizienten. Zur besseren Sichtbarkeit wurde das Beispiel 1 mit den Parametern $a = 2, b = -1, c = -1, d = -2$ und ohne Eingang verwendet.

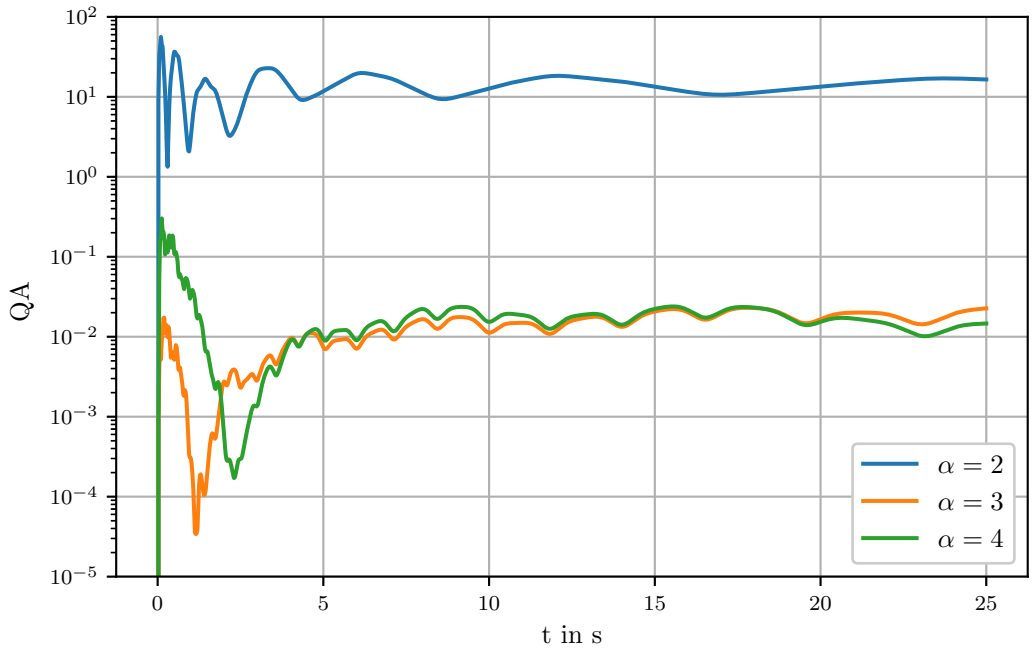

Abbildung 32 – QA bei unterschiedlichem Polynomgrad α. Es wurde das Beispiel 1 mit den oben genannten Parametern und mit Eingang verwendet. Bei Ansatzfunktionen höheren Grades kam es zu keiner Verbesserung des Ergebnisses.

Systembeschreibung ausreichen. Eine sinnvolle Vorgehensweise wäre, zuerst Systeme mit Ansatzfunktionen, die einen hohen Polynomgrad haben, zu identifizieren und nach und nach den Polynomgrad zu verringern, um so den minimalen Polynomgrad zu finden. Durch die Verkleinerung des Optimierungsproblems reduzieren sich die Abweichungen und beschleunigt sich die Berechnung.

3.3.4 Abhängigkeiten von ausgewählten Parametern beim ODEnet

Wie in Abschnitt 2.3.1 ausgeführt, hat ein KNN viele Parameter, die entscheidend für den Trainingserfolg sind. Auf einige besonders wichtige Parameter soll hier weiter eingegangen werden:

- die Lernrate,
- die Stichprobengröße,
- die Neuronenanzahl in jeder verborgenen Schicht,
- die Anzahl verborgener Schichten h,
- und den Optimierer.

Mit diesen Parametern besteht ein direkter Einfluss auf die Erstellung des KNNes hat. Neben diesen gibt es noch weitere, zum Beispiel die Initialisierung der Gewichte und Biase. Die Untersuchung aller Hyperparameter würde den Rahmen dieser Arbeit aber übersteigen.

Lernrate Die Lernrate steuert bei einer Veränderung der Gewichte, wie stark das Modell in Reaktion auf den geschätzten Fehler geändert werden soll. Damit ist dieser Hyperparameter entscheidend für den Lernerfolg und für die Geschwindigkeit des Trainings. Wird die Lernrate zu klein gewählt, verlängert sich der Trainingsprozess oder bleibt im schlechteste Fall hängen. Wird die Lernrate zu groß gewählt, ist es möglich, dass ein nicht optimaler Satz von Gewichten berechnet wird oder sogar ein instabiler Trainingsprozess entsteht. Insbesondere letzteres gilt es zu vermieden, da sonst das Training fehlschlägt.

Die Auswirkung der verschiedenen Lernraten auf das Testsystem ist Abbildung 33 zu entnehmen. Es ist ersichtlich, dass ab ca. 500 Epochen die Lernraten 10^{-3} und 10^{-2} einen ähnlichen Verlauf aufweisen. Am Anfang lernt das KNN mit der Lernrate 10^{-2} etwas schneller. Allerdings, gibt es bei dieser Lernrate zwei Ausreißer. Auch sieht die MQA etwas volatiler aus, als bei den beiden anderen Lernraten. Das lässt sich auf die größere Änderung der Gewichte zurückführen, die bei einer höheren Lernrate auftritt.

Lernrate	Trainingszeit in s
10^{-2}	1218
10^{-3}	1260
10^{-4}	1417

Tabelle 2 – Trainingszeiten bei unterschiedlichen Lernraten für das ODEnet

Die Volatilität des Signals nimmt also mit sinkender Lernrate ab.

Dass für eine Lernrate von 10^{-4} die MQA erst vergleichsweise langsam fällt, ist auf eine zu kleine Wahl der Lernrate zurückzuführen. Dafür gibt es keinen Ausreißer. Die Änderung der Gewichte erfolgt, wie oben bereits beschrieben, sehr langsam. Das wird auch anhand des Graphens deutlich. Um eine gleiche Genauigkeit zu erreichen wären also wesentlich mehr Epochen nötig, was aber zu einer längeren Trainingszeit führt.

Auch der Effekt auf die Trainingszeit lässt sich beobachten und ist in Tabelle 2 dargestellt. Mit einer Verkleinerung der Lernrate erhöht sich auch die Trainingszeit, wenn auch nur geringfügig.

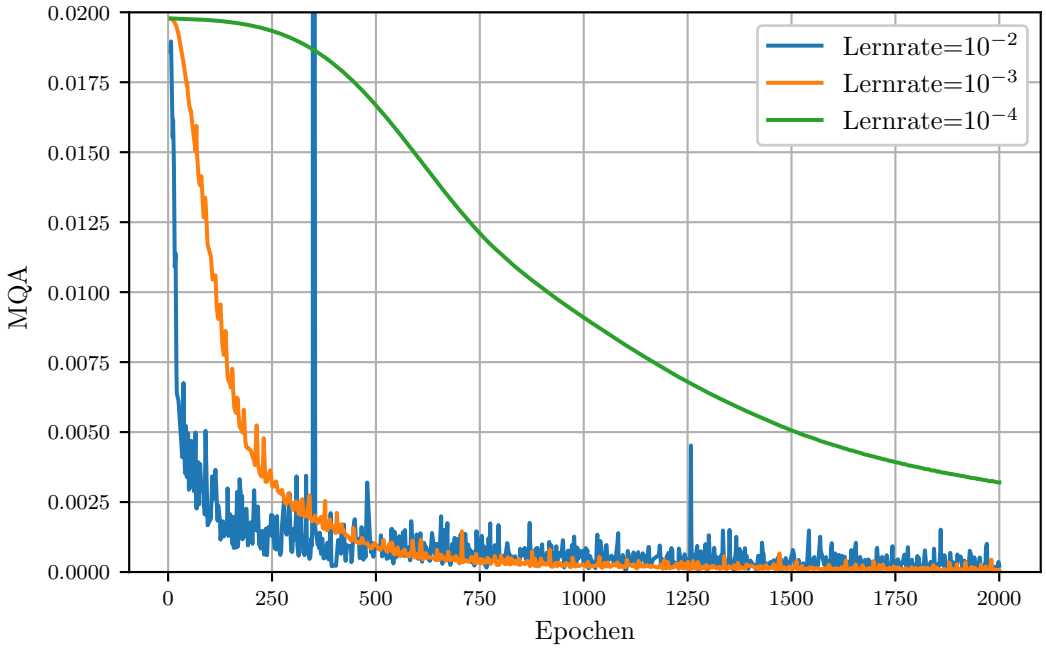

Abbildung 33 – MQA in Abhängigkeit von der Lernrate beim ODEnet

Stichprobengröße Wie die Lernrate hat die Stichprobengröße einen großen Einfluss auf den Erfolg und insbesondere auf die Trainingsgeschwindigkeit. Wie oben beschrieben, besteht eine Stichprobe aus einer Anzahl aufeinanderfolgender Datenpunkte. Es ist

anzunehmen, dass mit sich vergrößernder Stichprobengröße die Trainingszeit erheblich erhöht, weil ein größeres Datenpaket auf einmal verarbeitet werden muss.

In Abbildung 34 ist zu erkennen, dass mit größer werdender Stichprobengröße die MQA ansteigt. Auch nehmen die Anzahl und die Größe der Ausreißer zu. Dieses Verhalten lässt sich sehr gut durch das verwendete Testsystem erklären. Bei 25 aufeinanderfolgenden Datenpunkten liegt eine größere Änderung der Zustände vor, als bei einer kleineren Anzahl von Datenpunkten. Diesen Unterschied zu erlernen fällt dem KNN schwer. Das beste Ergebnis kann mit einer Stichprobengröße von zwei und fünf erreicht werden. Bei diesen Stichprobengrößen nimmt die MQA immer weiter ab.

Die Trainingszeiten verhalten sich, wie in Tabelle 3 aufgezeigt. Auffallend ist, dass bei einer Stichprobengröße von fünf, eine geringere Zeit benötigt wird als bei einer Stichprobengröße von zwei. Das lässt darauf schließen, dass es einen optimalen Punkt von Stichprobengröße und Trainingszeit gibt. Da der Unterschied aber nicht groß ist und sich bei einer Stichprobengröße von zehn die Trainingszeit schon vergrößert hat, wird davon abgesehen, die optimale Stichprobengröße zu finden.

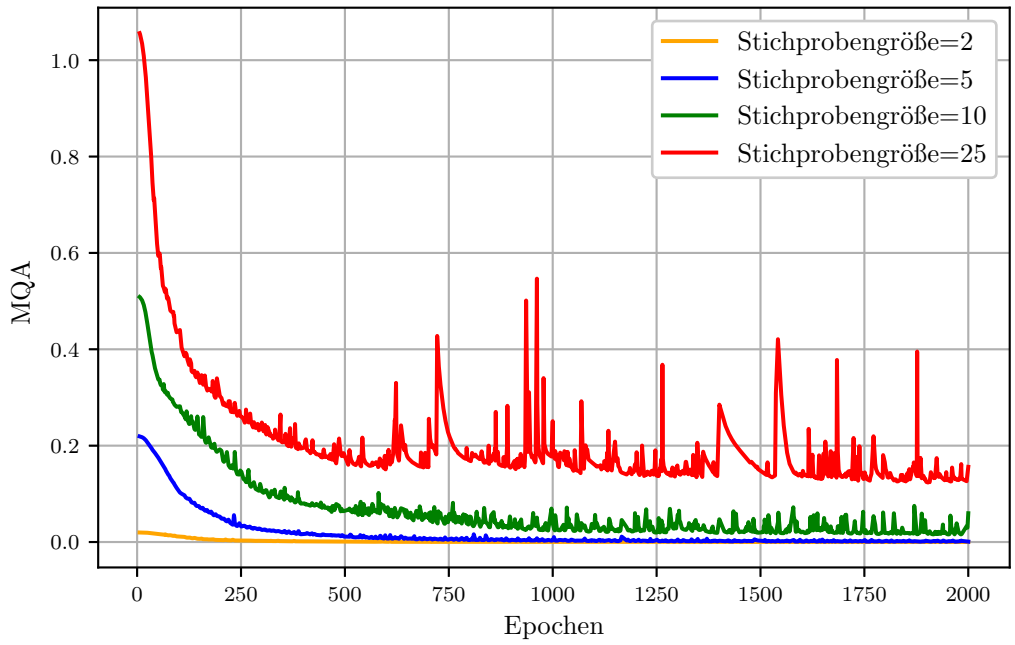

Abbildung 34 – MQA in Abhängigkeit von der Stichprobengröße beim ODEnet

Neuronenanzahl Die Neuronenanzahl ist ein entscheidender Faktor dafür, dass das KNN das System überhaupt lernen kann. Bei zu wenig Neuronen lernt das KNN das System nicht gut genug, bei zu vielen Neuronen verlängert sich die Trainingszeit,

Stichprobengröße	Trainingszeit in s
2	1260
5	1082
10	1332
25	3900

Tabelle 3 – Trainingszeiten bei unterschiedlichen Lernraten beim ODEnet

weil nicht alle Neuronen für das Finden der Systemfunktion benötigt, aber gleichwohl berechnet werden müssen. Wie sich die Neuronenanzahl auf das Training auswirkt ist in Abbildung 35 dargestellt. Es fällt auf, dass die MQA aller Graphen nach 2000 Epochen ca. gleich ist. Der einzige Unterschied ist die Anzahl der Epochen, die erforderlich ist, um die finale Genauigkeit zu erreichen. Die finale Genauigkeit wird mit 100 Neuronen nach ca. 400 Epochen erreicht. Im Gegensatz dazu braucht das KNN zum Erreichen des gleichen Wertes mit 40 Neuronen ca. 1300 Epochen.

Auch hier ist ein Vergleich der Trainingszeiten sinnvoll, denn diese unterscheiden sich erheblich, wie Tabelle 4 zu entnehmen ist. Mit steigender Neuronenanzahl erhöht sich die Trainingszeit erheblich. Auch ist der Einfluss wesentlich größer als der der Lernrate.

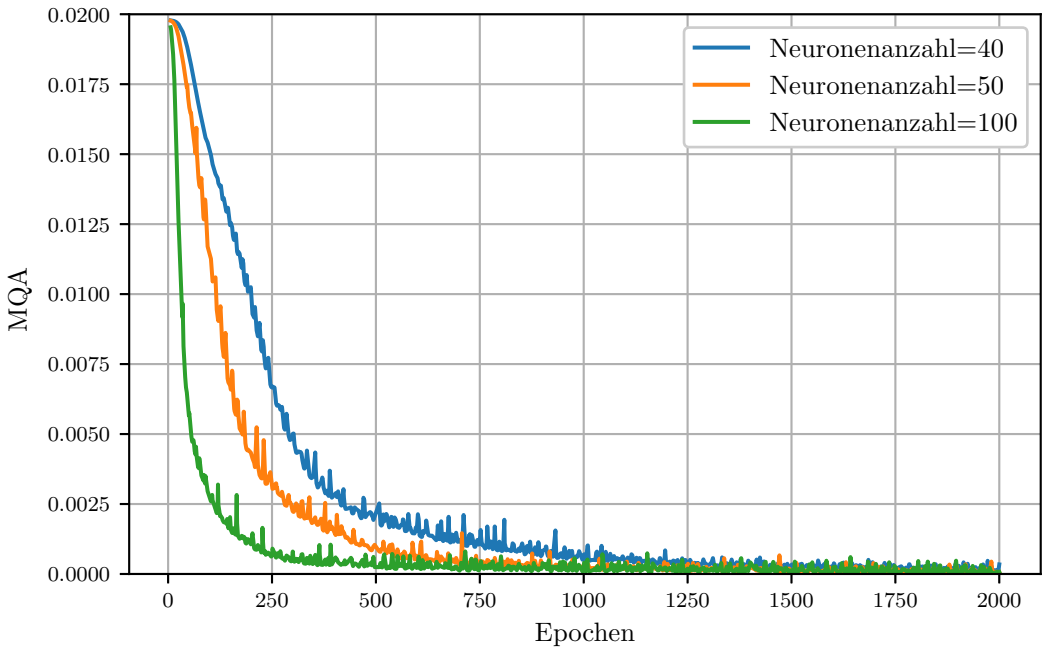

Abbildung 35 – MQA in Abhängigkeit von der Neuronenanzahl pro Schicht beim ODEnet

Anzahl Neuronen	Trainingszeit in s
40	968
50	1260
100	2018

Tabelle 4 – Trainingszeiten bei unterschiedlicher Neuronenanzahl pro Schicht beim ODEnet

Anzahl verdeckter Schichten Anhand der Anzahl der verdeckten Schichten kann beurteilt werden, ob der Einfluss der Schichten oder die Neuronenanzahl stärker wiegt. In Abbildung 36 ist der Verlauf bei verschiedenen Anzahlen verdeckter Schichten aufgetragen. Festzustellen ist, dass die MQA für drei verdeckte Schichten schlechter ist, als bei zwei und vier verdeckten Schichten. Die Gesamtanzahl der Neuronen blieb dabei immer gleich. Die MQA für zwei und vier verdeckte Schichten ist fast gleich, was in dem Sinne überraschend ist, weil eine Verschlechterung zu bei 2 verdeckten Schichten zu erwarten gewesen wäre. Gleichwohl wird bei allen drei Versuchen die minimale MQA erreicht, wenn auch nach unterschiedliche langen Trainingszeiten.

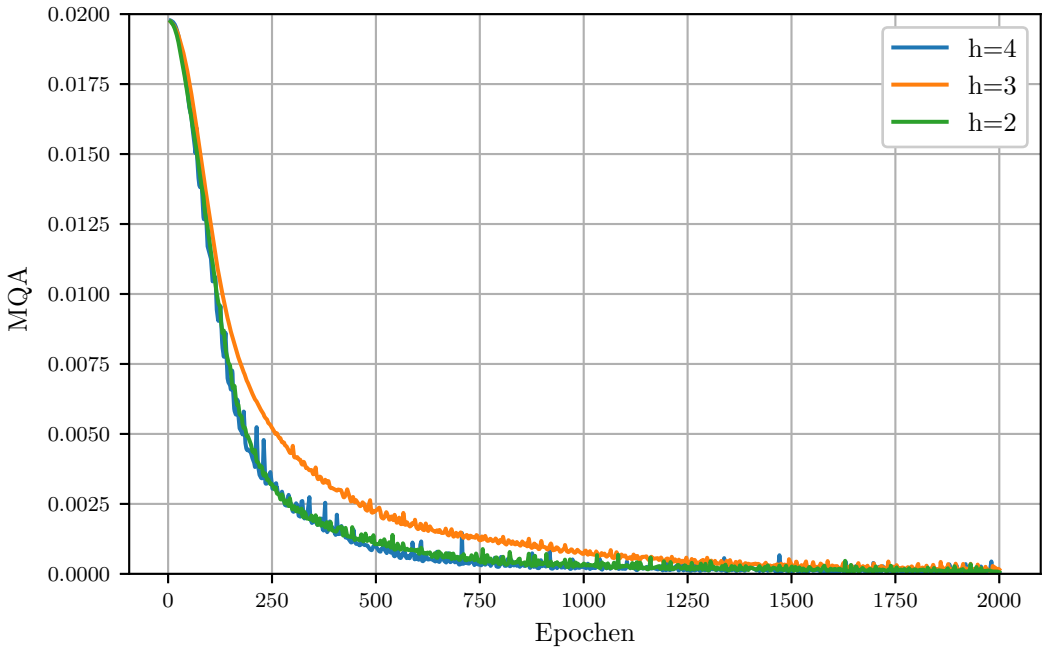

Abbildung 36 – MQA in Abhängigkeit von der Anzahl verdeckter Schichten h beim ODEnet

Optimierer Interessant ist auch das Ergebnis verschiedener Optimierer. Keras stellt dafür eine Klasse zur Verfügung, in der verschiedene Optimierer implementiert sind. Für

die Art von Training, das hier durchgeführt wird, kommen die Optimierer *RMSprop*, *Adam* und *Adamax* in Frage. Die Verläufe der MQA für die drei vorgenannten Optimierer sind in Abbildung 37 gezeigt. RMSprop und Adam zeigen, bis auf die ersten 500 Epochen, ein sehr ähnliches Verhalten, wohingegen Adamax sich vergleichsweise langsam an die minimale MQA annähert und diese auch nach 2000 Epochen noch nicht vollständig erreicht. Weiterhin fällt auf, dass es beim Adam-Optimierer immer wieder zu kleinen Ausreißern kommt. Auch bei RMSprop zeigt sich dieses Verhalten gelegentlich. Dagegen tritt es bei Adamax nicht auf. Da die Optimierer eine integrierte Schrittweitensteuerung haben ist davon auszugehen, dass an diesen Punkten die Schrittweite leicht erhöht wurde und damit die Abweichung steigt. Entscheidend für den Erfolg des Trainings sind die Ausreißer nicht, dann im Mittel sinkt die MQA.

Bezüglich der Trainingszeit liegen die drei getesteten Optimierer in einer kleinen Zeitspanne. Sie haben keinen entscheidenden Einfluss auf die Länge des Trainings, wie andere oben schon genannte Parameter des KNN. Dies ist Tabelle 5 zu entnehmen.

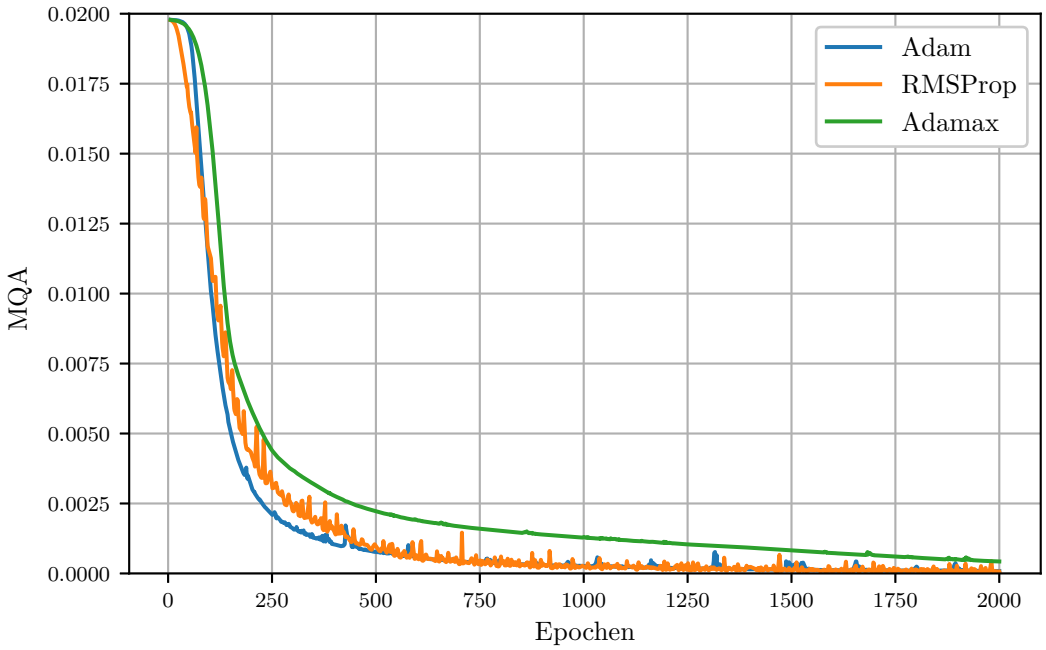

Abbildung 37 – MQA in Abhängigkeit vom Optimierer beim ODEnet

Optimierer	Trainingszeit in s
RMSprop	1260
Adam	1160
Adamax	1197

Tabelle 5 – Trainingszeiten bei unterschiedlichen Optimierern beim ODEnet

	Versuch 1	Versuch 2	Versuch 3
SINDYc	19 s	250 s	241 s
ODEnet	8385 s	7183 s	6815 s

Tabelle 6 – Vergleich der Trainingszeiten von SINDYc und ODEnet

3.4 Vergleich der Systemidentifikationsansätze

3.4.1 Vergleich der Ergebnisse der Testsysteme

Um eine Entscheidung dafür treffen zu können, welcher der beiden Ansätze die größere Erfolgsaussicht hat, werden die Ergebnisse der beiden Ansätze für zwei ausgewählte Versuche untersucht. Das wichtigste Maß für eine Entscheidung ist die MQA. Jedoch soll auch die Trainingszeit in Betracht gezogen werden.

Zum Vergleich werden die im Abschnitt 3.3.2 vorgestellten Versuche genutzt, da die KNN schon trainiert sind und die Berechnung mit SINDYc wesentlich schneller ist. Die Datensets erlernt nun auch SINDYc.

Generell kann für alle getesteten Versuche gesagt werden, dass die Trainingszeit für den ODEnet-Ansatz weitaus größer ist, als beim SINDYc-Ansatz. Ursache hierfür ist die langwierigen Optimierung, die während des KNN-Trainings durchgeführt werden muss. Sie entfällt beim SINDYc-Ansatz, da hier lediglich ein mathematisches Optimierungsproblem zu lösen ist. Bei dem VGF-Kristallzüchtungsprozess hat die Trainingszeit keine entscheidende Bedeutung, da die Systemgleichungen gefunden werden sollen, was offline erfolgen kann und nicht online während des Prozesses. Der Vergleich der Trainingszeiten für die drei durchgeführten Versuche ist in Tabelle 6 aufgezeigt. Beim Training des ODEnet beträgt die Trainingszeit zwei Stunden oder länger, wohingegen die Trainingszeit von SINDYc nur wenige Minuten beträgt. Die Trainingszeit bei SINDYc kann durch die Verwendung anderer Bibliotheken noch erheblich beschleunigt werden. Die Implementierung der LASSO-Regression gestaltet sich dann schwieriger oder ist gar unmöglich. Jedoch ist die LASSO-Regression für die Genauigkeit der Systemidentifikation nicht ausschlaggebend, sondern nur für die dünne Besetztheit.

Demzufolge ist das ausschlaggebende Kriterium die Abweichung von den Testdaten. Auch bei diesem Test schneidet der SINDYc-Ansatz wesentlich besser ab als das ODEnet. Dies zeigen die Tabellen 7 und 8. Die sehr großen Abweichungen hängen auch mit den

	Versuch 1	Versuch 2	Versuch 3
SINDYc	14,1	17 223,0	207 488,0
ODEnet	1796,0	3 698 541,0	868 862,0

Tabelle 7 – Vergleich der MQA von SINDYc und ODEnet

	Versuch 1	Versuch 2	Versuch 3
SINDYc	0,014	17,2	207,5
ODEnet	1,8	3698,5	868,9

Tabelle 8 – Vergleich der MQA pro Datenpunkt von SINDYc und ODEnet

gewählten Parametern der Testsysteme zusammen. Gleichwohl ist zu erkennen, dass SINDYc im Vergleich zum ODEnet wesentlich kleinere Abweichungen aufweist. Das kann auch bei einer einfachen Betrachtung der gefundenen Phasenportraits in den Abbildungen 38, 39 und 40 beobachtet werden. Die Abweichungen von SINDYc sind noch weiter reduzierbar, indem die Anzahl der Datenpunkte vergrößert wird. Dieser Effekt wurde bereits in Abschnitt 3.3.3 gezeigt. Jedoch sollten an dieser Stelle die beiden Ansätze verglichen werden, weswegen gleiche Datensets benutzt wurden. Es sind hier also keine optimalen Ergebnisse von SINDYc dargestellt, weil das Training des ODEnet sonst zu lange gedauert hätte.

Es fällt weiterhin auf, dass bei einem Datenset aus verschiedenen Eingängen es das ODEnet nicht schafft, das System richtig darzustellen. Demgegenüber löst SINDYc dieses Problem besser.

3.4.2 Zusammenfassung der Erkenntnisse der Ansätze mit den Testsystemen

Tabelle 9 soll die gewonnen Erkenntnisse aus den Versuchen mit den Testsystemen noch einmal anschaulich darstellen.

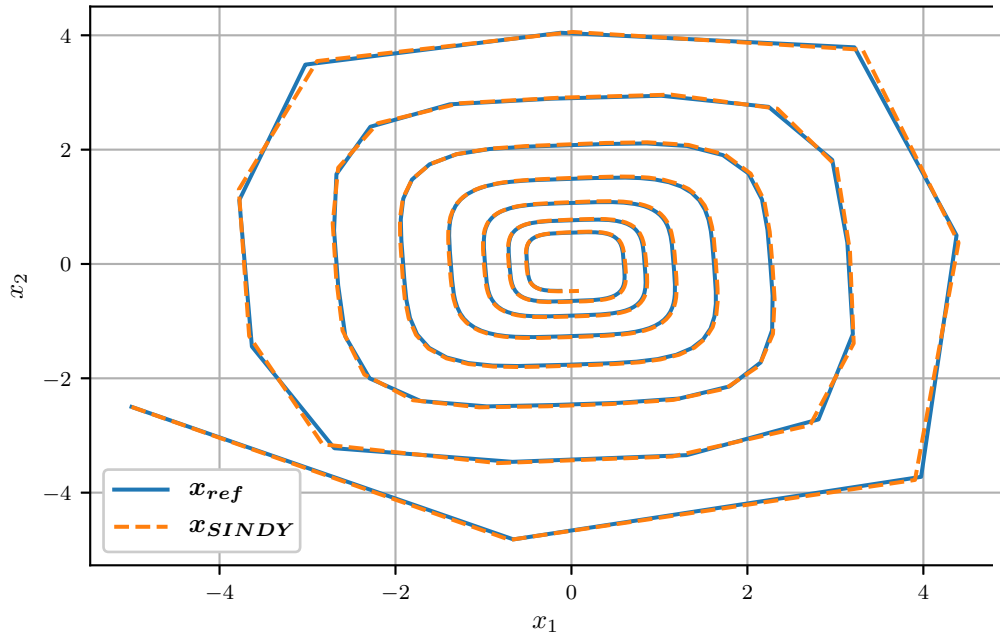

Abbildung 38 – Phasenportrait für SINDYc für das Beispiel 1 ohne Eingang und Anfangswertintervall $x_0 = [-5, 5]$ und zehn Datensets

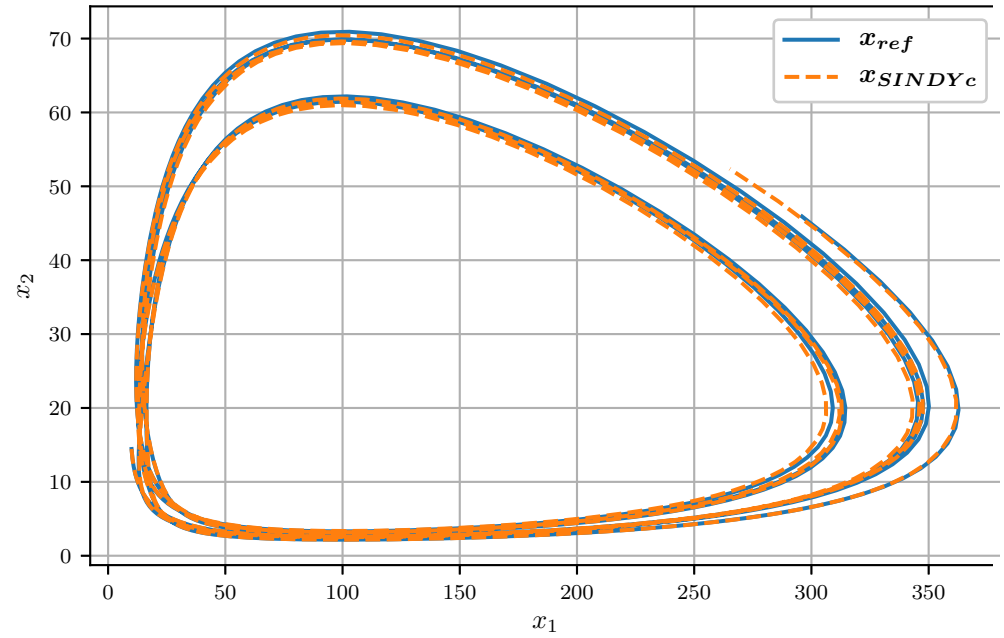

Abbildung 39 – Phasenportrait für SINDYc für das Beispiel 2 mit zwei unterschiedlichen Sinus-Eingängen und Anfangswertintervall $x_0 = [10, 50]$ und zehn Datensets

	SINDYc	ODEnet
Ergebnisqualität bei Systemen ohne Eingang	Nahezu perfekt, sobald ausreichend Trainingsdatenpunkte vorhanden sind. Bei nichtlinearen Systemen kommt es zu Abweichungen, weil die Koeffizienten der Ansatzfunktionen nicht genau gefunden werden, sondern mit minimalen Abweichungen. Der Grund dafür ist die LASSO-Regression.	Schlechter als bei SINDYc. Insbesondere bei großen Zeiten kommt es zu signifikanten Abweichungen.
Ergebnisqualität bei Systemen mit Eingang	Sehr gut, sofern ausreichend Trainingsdatenpunkte vorhanden sind. Etwas schlechter als ohne Eingang. Die Stärke stellt hier das gute Verhalten bei verschiedenen Eingängen dar.	Ist wesentlich schlechter als SINDYc. Das Ergebnis bei einem trainierten Eingang in Ordnung. Sobald die Trainingseingänge erheblich voneinander abweichen, sind die Ergebnisse nicht mehr brauchbar. Die Systemidentifikation schlägt an dieser Stelle fehl.
Trainingszeit	Kurz (wenige Minuten). Sie kann durch die Nutzung von Optimierungsbibliotheken für Python noch weiter reduziert werden. Erhöht sich bei mehr Ansatzfunktionen und mehr Datenpunkten.	Erheblich (mehrere Stunden). Sie ist abhängig von der Neuronenanzahl pro Schicht, der Anzahl der verdeckten Schichten und dem Optimierer.
Sensitivität auf Hyperparameteränderungen	Entscheidend ist die Auswahl der gewählten Ansatzfunktionen. Falsche Ansatzfunktionen führen zu einem schlechten Ergebnis. Andere Hyperparameter haben eher einen marginalen Einfluss und sind einfach, jedoch für jedes System unterschiedlich, einzustellen.	Bessere Ergebnisse mit einer höheren Neuronenanzahl pro Schicht und Anzahl verdeckter Schichten, was zu einer Trainingszeitverlängerung führt. Die Veränderung der Lernrate führt zu einem besseren Ergebnis. Allerdings besteht auch hier ein längere Trainingszeit und das Risiko, ein lokales Minimum zu erreichen.
Ergebnis bei Daten mit Rauschen	Gut, aber mit sich erhöhender Standardabweichung schlechter werdend. Die Daten müssen gegebenenfalls gefiltert werden.	Schlechter als bei SINDYc. Erheblich Abweichungen bestehen bereits bei geringem Rauschen. Auch hier sollten die Daten gefiltert werden.
Nötiges Vorwissen über das zu identifizierende System	Ungefähres Wissen über enthaltene Funktionen ist für die Wahl der Ansatzfunktionen hilfreich, aber nicht unbedingt erforderlich.	Kein Vorwissen nötig

Tabelle 9 – Zusammenfassung der Ergebnisse der Testsysteme

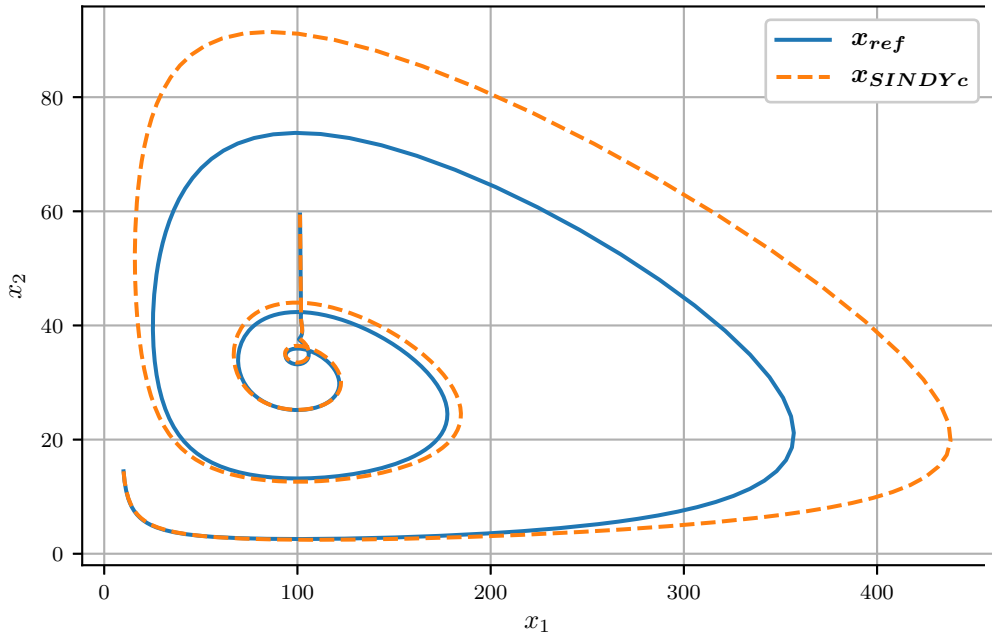

Abbildung 40 – Phasenportrait für SINDYc für das Beispiel 2 mit zwei verschiedenen Eingängen und Anfangswertintervall $x_0 = [10, 50]$ und zehn Datensets

Kapitel 4

Tests mit Anlagendaten

Nach den Versuchen mit den Testsystemen, sind reale Systeme zu identifizieren. Dafür werden Daten des VGF-Kristallzüchtungsprozess bereitgestellt, die von den beschriebenen Systemidentifikationsansätze gelernt werden sollen. Hierbei sollen die Erkenntnisse der Tests mit den Testsystemen möglichst umfassend genutzt werden. Die Implementierung der beiden Ansätze soll im Kern erhalten bleiben und nur an den Stellen Dateneingabe, Datenvorverarbeitung und Visualisierung verändert werden, damit auch ein Vergleich mit den Ergebnissen der Testsysteme möglich ist. Außerdem soll eine systematische Näherung an ein gutes Ergebnis erfolgen.

4.1 Beschreibung der Daten

Als Datengrundlage sollen Züchtungsdaten im VGF-Kristallzüchtungsprozess dienen. Wie in Abschnitt 2.1 beschrieben, ist der Temperaturverlauf im Medium entscheidend für die Güte des Züchtungsprozesses. Für die Tests mit realen Anlagendaten stehen zwei verschiedene Prozesse zur Verfügung. Aus diesen Daten sollen zwei unterschiedliche Systeme identifiziert werden.

4.1.1 Datenbasis

Erster Prozess Der erste Prozess besteht aus zwei Trainingsdatensätzen und einem Testdatensatz, wobei für die Systemidentifikation nur die zwei Trainingsdatensätze bekannt sind. Der unbekannte Testdatensatz soll nach dem Training zur Verifizierung der Systemidentifikation fungieren. Im ersten Trainingsdatensatz wurde das Temperaturverhalten am Tiegel bei einer bestimmten Trajektorie der Heizer gemessen. Im zweiten Trainingsdatensatz wurden Sprungantworten der Heizer und deren Auswirkung auf die Temperatur im Medium aufgenommen. Die Trainingsdatensätze bestehen aus Zeitverläufen mit 55440 bzw. 42711 Zeitpunkten. Der Prozess ist äquidistant abgetastet. Von den fünf Heizelementen sind die zugehörigen Zeitverläufe der Leistung gegeben.

Diese können als die Eingänge $u_i, i = 1, \ldots, 5$ betrachtet werden. Weiterhin sind die Temperaturen an den Heizelementen bekannt. Das sind die Zustände $x_{\text{ref},j}, j = 1, \ldots, 5$. Aufgabe ist, den Zusammenhang zwischen Heizleistung und Temperaturverlauf zu finden.

Die Eingänge des ersten Trainingsdatensatzes sind in Abbildung 41 dargestellt. Wie oben beschrieben wird mit diesen eine Trajektorie abgefahren. Die drei Mantelheizer $2, \ldots, 4$ zeigen ein gleiches Verhalten, wohingegen die zwei Stirnseitenheizer 1 und 5 ab der Zeit $t = 1$ ungefähr gegensätzliche Leistungen haben. Die Schmelze wird mittels der drei Mantelheizern lange auf einer Temperatur gehalten, mit einem nur leichten Abfall bei u_4. Entscheidend für den Identifikationserfolg wird auch sein, wie gut die Ansätze die schnellen Anstiege bis $t = 0{,}3$ lernen.

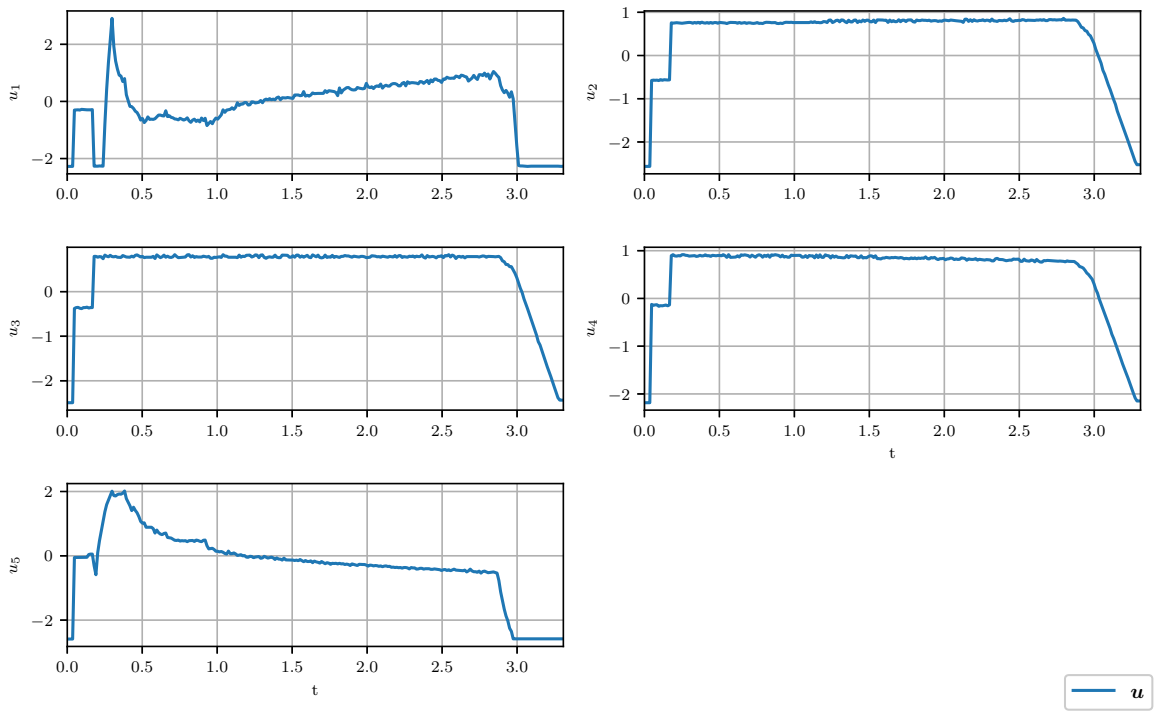

Abbildung 41 – Eingangskomponenten des ersten Trainingsdatensatzes, des ersten Prozesses

In Abbildung 42 sind die Zustände des ersten Trainingsdatensatzes des ersten Prozesses dargestellt. Die Zustände zeigen alle ein sehr ähnliches Verhalten und unterscheiden sich nur in ihren Amplituden signifikant. Entscheidend für die Systemidentifikation ist, dass die Änderung von concav zu convex bei $t = 0{,}9$ gelernt wird. Diese Umkehrung rührt von einem Sprung in den Eingängen u_1 und u_5 her. Aufgrund der sehr langsamen Änderungen über die Zeit, ist es möglich, dass für diesen Trainingsdatensatz Ansatzfunktionen ersten Grades ausreichen. Es ist aber ebenso möglich, dass mit Ansatzfunktionen zweiten Grades bessere Ergebnisse erzielt werden.

Beim Betrachten der Eingänge des zweiten Trainingsdatensatzes in Abbildung 43,

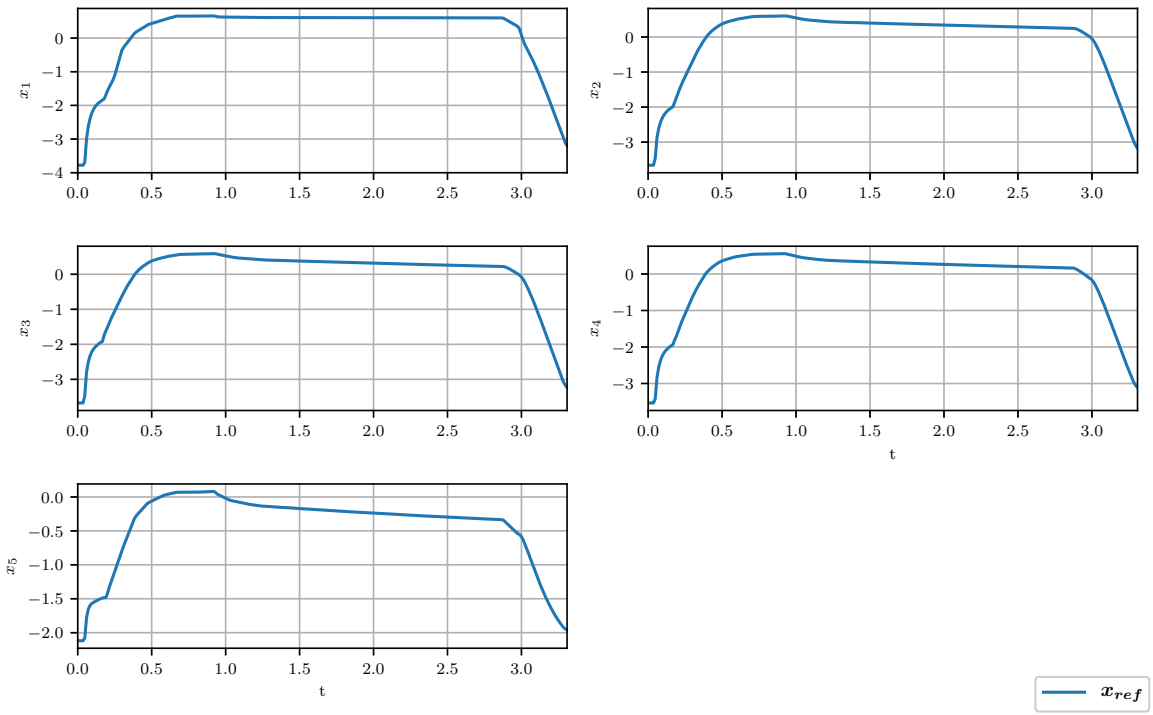

Abbildung 42 – Zustandskomponenten des ersten Trainingsdatensatzes, des ersten Prozesses

zeigt sich der Unterschied zu Trainingsdatensatz 1. Es erfolgen Leistungssprünge zu verschiedenen Zeitpunkten an den Mantelheizern und am Stirnseitenheizer 5. Durch die Sprünge zu verschiedenen Zeitpunkten soll untersucht werden, wie sich ein Sprung auf das gesamte System auswirkt.

Die zugehörigen Zustände sind in Abbildung 44 dargestellt. Anfänglich erfolgt eine Abkühlung, worauf drei deutlich sichtbare Phasen einer Aufheizung und leichten Abkühlung erfolgen. Diese Phasen haben jeweils die Form einer logarithmisch steigenden, bzw. exponentiell fallenden Funktion, oder die typische Lade- und Entladekurven eines Kondensators (PT1-Verhalten). Deswegen wird es ausreichend sein, lineare Funktionen für SINDYc anzusetzen, denn das System ist linear.

Für das Identifizieren des Systems ist es erforderlich, dass bei beiden Trainingsdatensätze die gleichen Koeffizienten von SINDYc gefunden werden. Das ist aufgrund der erheblichen Unterschiede der Trainingsdatensätze schwierig. Deswegen kann es erforderlich sein Ansatzfunktionen zweiten Grades zur Systemidentifikation zu verwenden.

An dieser Stelle kann das Ergebnis des ODEnetes besser sein, weil es zur Optimierung mehr Freiheitsgrade zur Verfügung hat. Als Anfangsparameter für das ODEnet werden die Parameter der Testsysteme verwendet werden. Aufbauend auf den Erkenntnissen aus Abschnitt 3.3.4, werden die Anzahl der verdeckten Schichten, die Anzahl der Neuronen

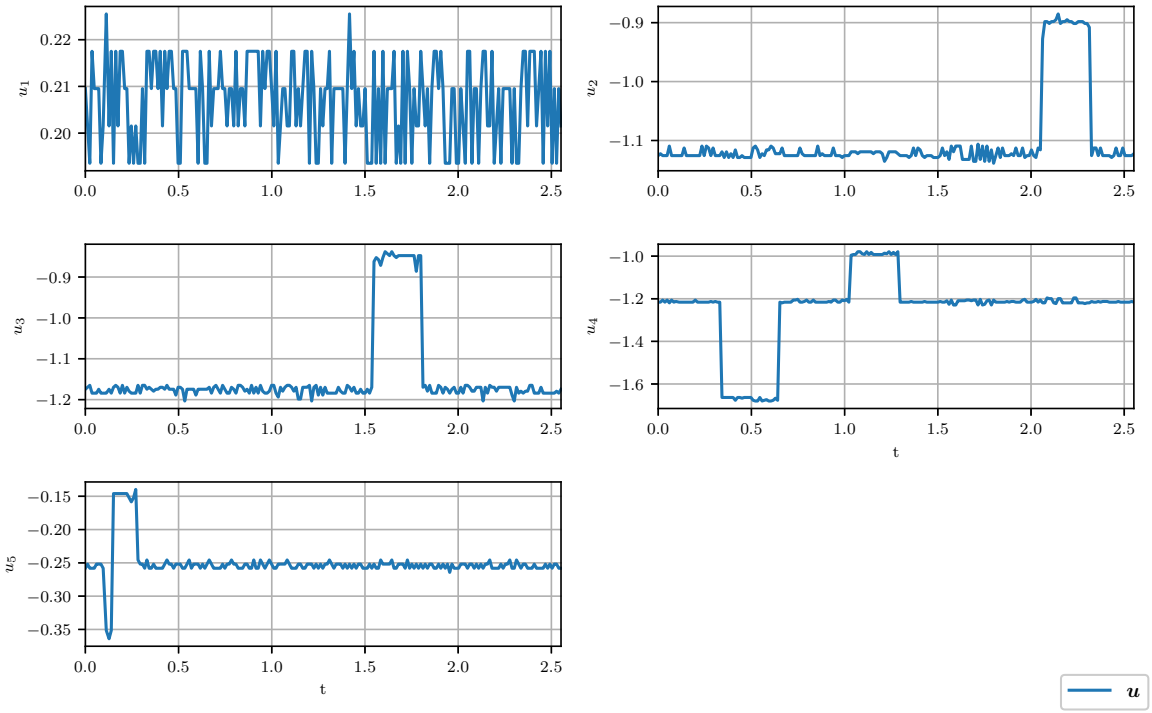

Abbildung 43 – Eingangskomponenten des zweiten Trainingsdatensatzes, des ersten Prozesses

pro Schicht und die Lernrate sukzessive verändert, bis ein optimales Ergebnis erreicht wird.

Zweiter Prozess Der zweite Prozess besteht aus einem Trainingsdatensatz und einem Testdatensatz. Der Trainingsdatensatz enthält 1800 Messungen, wohingegen der Testdatensatz aus 200 Messungen besteht. Jede Messung besteht aus 99 Datenpunkten, die während eines Züchtungsprozesses aufgenommen wurden. Der Testdatensatz wird für die Verifizierung der Systemidentifikation nicht für das Training genutzt. Der erste Unterschied zum ersten Prozess besteht darin, dass es lediglich Informationen der an den beiden Stirnseiten gelegenen Heizern u_1 und u_5 gibt. Der zweite Unterschied ist, dass es einen zusätzlichen Zustand $x_{\text{ref},6}$ gibt. Dieser beschriebt die z-Position der Phasengrenze (Übergang von Schmelze zu Kristall, siehe Abbildung 1). Die fünf Temperaturen wurden erneut an den Heizern gemessen. Auch sind bei diesem Datensatz keine Heizleistungen, sondern Heiztemperaturen gegeben. Ziel ist es, anhand der zwei beiden Eingänge die fünf Temperaturzustände und die Phasengrenze zu identifizieren.

Abbildung 45 zeigt, dass die Eingänge linear sind. Das ist der Fall für alle 1800 Messungen, die sich aber in ihren Anstiegen, Start- und Endwerten unterscheiden. Eine Gesamtübersicht über die Eingangssignale zu stets dem selben Zeitpunkt ist in Abbildung 46 dargestellt. Anhand dieser Darstellung wird deutlich, dass die Eingänge über die

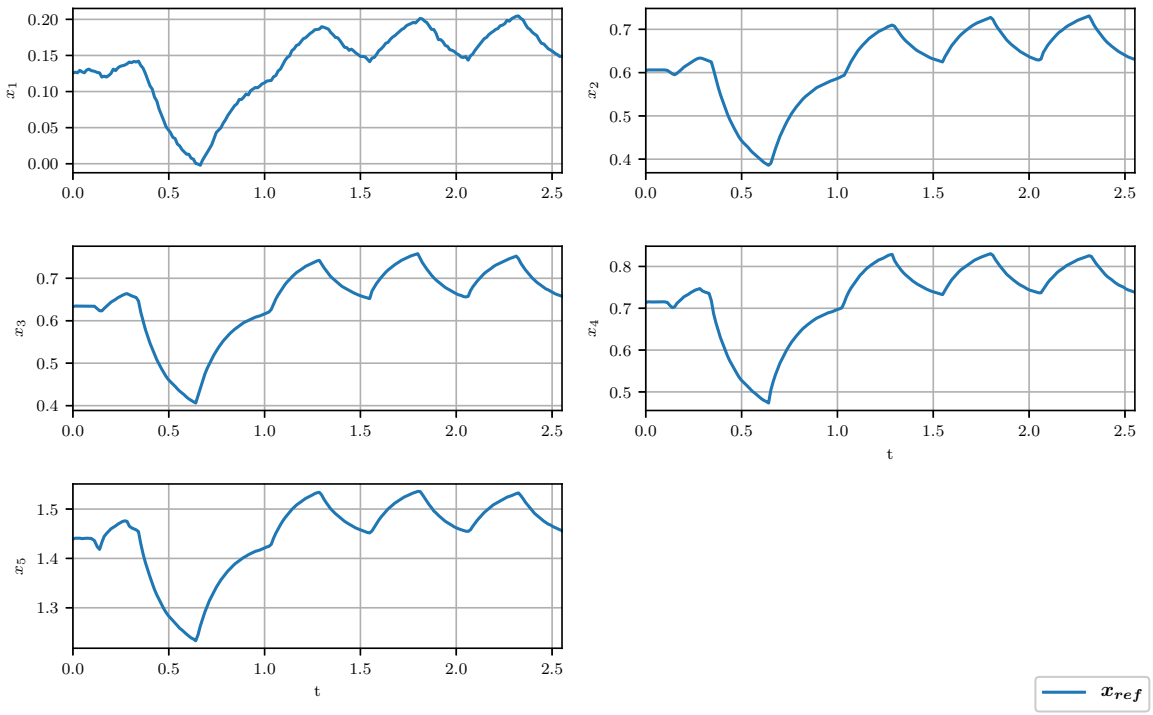

Abbildung 44 – Zustandskomponenten des zweiten Trainingsdatensatz, des ersten Prozesses

Anzahl der Messungen ähnlich sind, sich aber in ihren Temperaturwerten unterscheiden. Diese Erkenntnis ist wichtig für den SINDYc-Ansatz, der bei einer größeren Ähnlichkeit der Eingänge, das eigentliche Systemverhalten besser lernt, nicht jedoch den Einfluss der Eingänge. Dabei steigt auch das Risiko, dass die Eingänge verwechselt werden, was zur Berechnung eines falschen Ergebnisses führen würde.

In Abbildung 47 sind die Zustände des zweiten Prozesses aufgetragen. Da der Datensatz aus vielen Messungen besteht ist es nicht möglich hier alle zu zeigen. Generell zeigen die Verläufe aber alle ungefähr das Verhalten, wie es in Abbildung 47 dargestellt ist. Das System ist nichtlinear, auch wenn man aufgrund der linearen Verläufe der Zustandstrajektorien auf die Linearität des Systems schließen könnte. Es ist auffällig, dass die Temperaturzustände $x_{\text{ref},1}$ und $x_{\text{ref},5}$, sowie die Phasengrenze einen linearen Verlauf aufweisen, wohingegen die Temperaturzustände $x_{\text{ref},2}, \ldots, x_{\text{ref},4}$. Das nichtlineare Verhalten entsteht durch das Überschreiten der Phasengrenze und die damit verbundene schnellere Abkühlung des Kristalls im Gegensatz zur Schmelze. Es gilt also herauszufinden, welcher der beiden Ansätze das nichtlineare Verhalten am besten identifizieren kann.

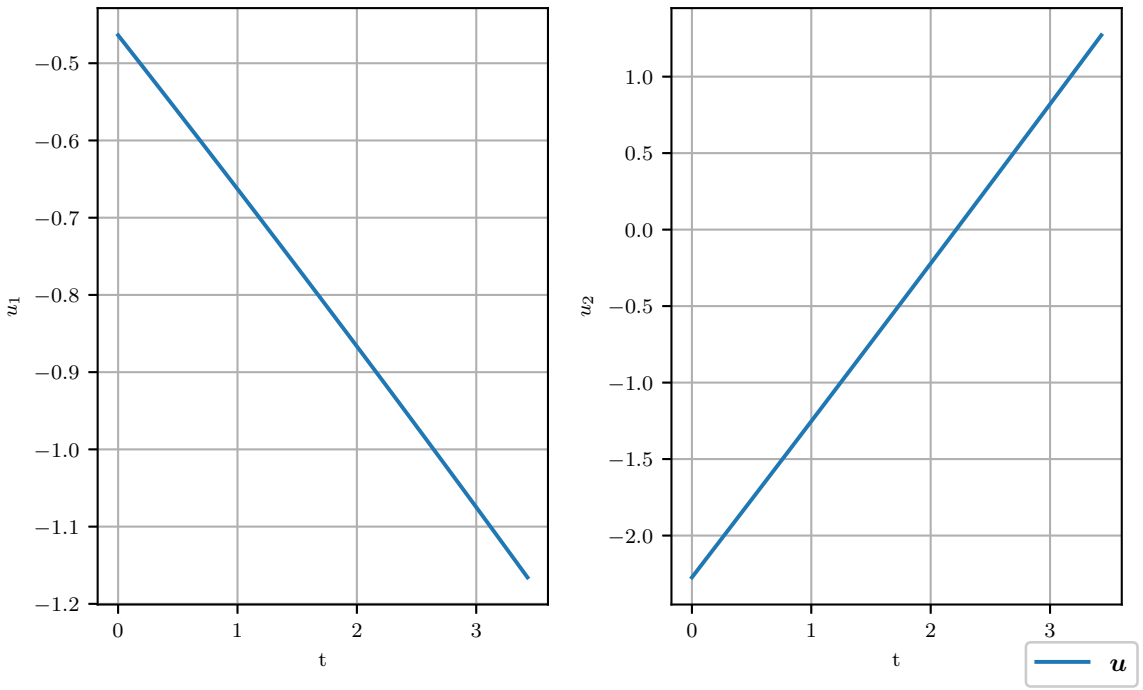

Abbildung 45 – Eingangskomponenten für eine Messung des zweiten Prozesses

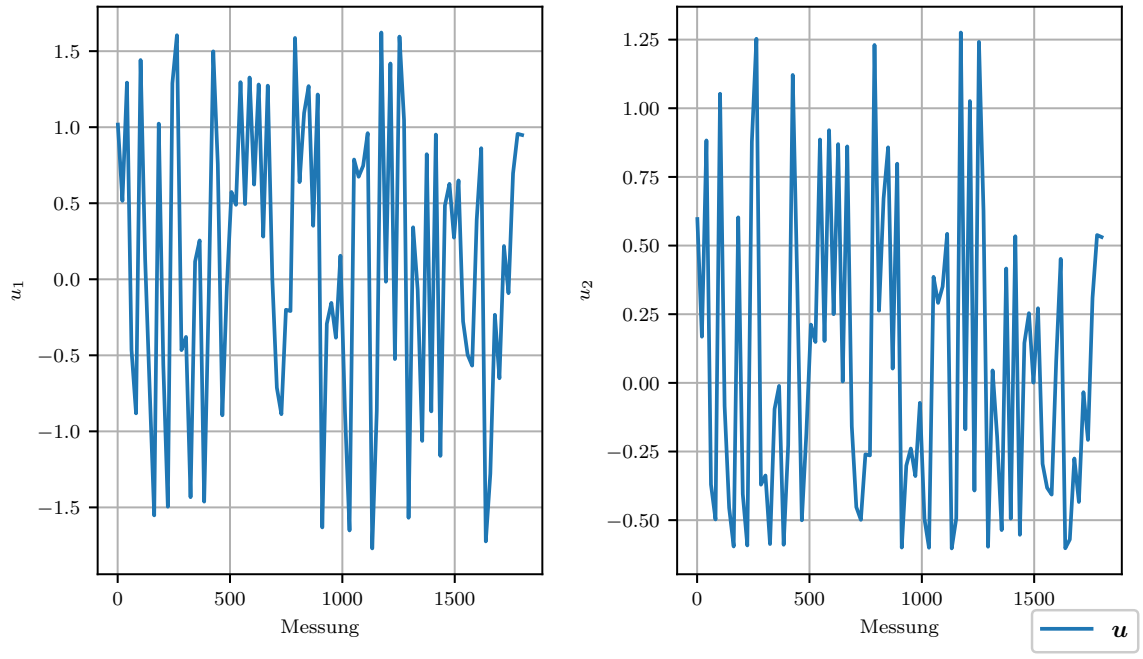

Abbildung 46 – Eingangskomponenten für alle Messungen des zweiten Prozesses an einem Zeitpunkt

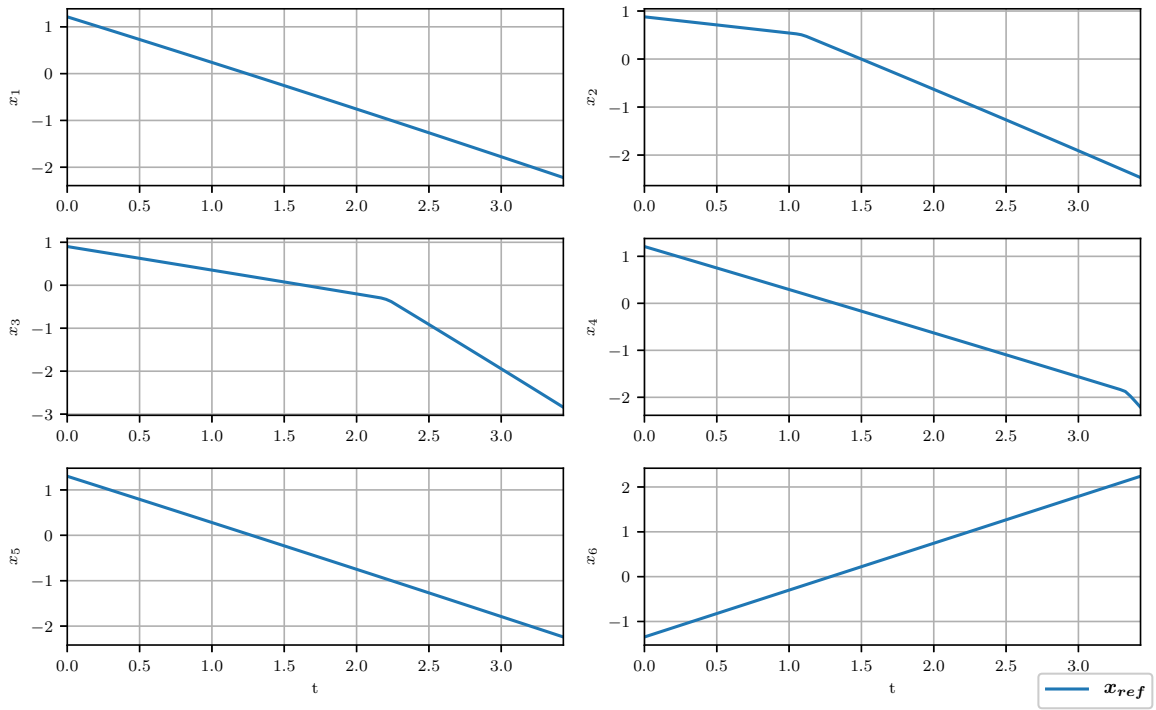

Abbildung 47 – Zustandskomponenten für eine Messung des zweiten Prozesses. Das nichtlineare Verhalten entsteht durch das Überschreiten der Phasengrenze von Schmelze zu Kristall und der damit verbundenen schnelleren Abkühlung.

4.1.2 Vorverarbeitung der Daten

Bevor die Daten von den beiden Ansätzen SINDYc und ODEnet gelernt werden, sind Vorverarbeitungen erforderlich. Da die Daten bereits standardisiert übergeben werden, erübrigt sich eine erneute Standardisierung. Da auch die Zeit standardisiert wurde, ist eine Verschiebung aller Werte um den kleinsten Wert nötig, weil der Prozess sonst nicht bei $t = 0$ beginnen würde. Das ist jedoch für das Funktionieren der Differentialgleichungslöser erforderlich und vereinfacht die Darstellung. Demzufolge gilt

$$\boldsymbol{t}_{\text{neu}} = \boldsymbol{t}_{\text{alt}} + |\min \boldsymbol{t}_{\text{alt}}|. \tag{4.1}$$

Das ist möglich, weil $\min \boldsymbol{t}_{\text{alt}} < 0$.

Beim ersten Prozess wird eine weitere Maßnahme ergriffen, die sich aus der großen Menge der Datenpunkte und der geringen Änderung von einem zum nächsten Datenpunkt bedingt: Es werden beim Lernen nicht alle Datenpunkte benutzt, sondern nur jeder r-te. Wird r gut gewählt, resultiert das in einer signifikanten Verringerung der Trainingszeit ohne eine Erhöhung der Abweichung von der Referenz. Hier wurde $r = 200$ gewählt. Das entspricht 1/277-el der Grundgesamtheit (55440 Datenpunkte).

Durch die Verwendung eines Differentialgleichungslöser mit variabler Schrittweite bei SINDYc, ist es erforderlich die Eingänge $\boldsymbol{u}$ zu interpolieren. Es wird eine Interpolation 1. Ordnung durchgeführt.

4.2 Tests der Prozesse mit SINDYc

4.2.1 Prozess 1

Beim Test von SINDYc des Prozess 1 werden zuerst die Trainingsdatensätze einzeln getestet, um zu evaluieren, ob überhaupt eine Identifikation des Systems möglich ist. Daraufhin werden beide Datensätze von SINDYc gelernt und getestet, ob auch eine Kombination der Trainingsdatensätze möglich ist, oder ob SINDYc dazu nicht in der Lage ist.

Für die Identifikation mit SINDYc werden $\lambda = 1$ und $D = 10^1$ genutzt. Es werden ausschließlich Polynome als Ansatzfunktionen genutzt und der Ansatzfunktionengrad variiert zwischen den einzelnen Prozessen.

Einzelidentifikation mit dem ersten Trainingsdatensatz Für den Test mit dem ersten Trainingsdatensatz wurden Ansatzfunktionen ersten Grades benutzt. Außerdem wurden am Anfang λ sehr niedrig und D so gewählt, dass keine Koeffizienten 0 gesetzt werden. Das Ergebnis ist in Abbildung 48 dargestellt. Wie angenommen schafft es

SINDYc, das System zu identifizieren. Es gibt nur zu Beginn kleine Abweichungen. Die Trainingszeit beträgt 3,7 s.

Im nächsten Schritt wird untersucht, ob sich das Ergebnis bei einer Erhöhung des Grades der Ansatzfunktionen verbessert. Dafür wird das gleiche Training mit Ansatzfunktionen zweiten Grades durchgeführt. Das Ergebnis ist auch in Abbildung 48 dargestellt und nicht gut. SINDYc schafft es nicht, die Koeffizienten richtig zu finden, was zu hohen Abweichungen führt, wie es Abbildung 49 zeigt. Diese liegen für $\alpha = 2$ mehrere Größenordnungen über denen für $\alpha = 1$. Ein Grund dafür kann sein, dass die LASSO-Regression ein lokales Minimum findet, das nicht für alle Trainingsdaten richtig ist. Im Fall dieses Trainingsdatensatzes sind eindeutig die Ansatzfunktionen ersten Grades zu bevorzugen. Anhand von Abbildung 49 lässt sich des Weiteren erkennen, dass insbesondere bei Änderungen der Trajektorie, wie es bei $t = 3$ der Fall ist, Änderungen in der QA auftreten. Während für Ansatzfunktionen ersten Grades die QA an dieser Stelle erheblich steigt und während des darauf folgenden linearen Abschnitts wieder fällt, zeigt die QA für Ansatzfunktionen zweiten Grades genau gegensätzliches Verhalten. An der Stelle des Übergangs von sinkt die QA, sobald die Trajektorie aber wieder in den linearen Verlauf übergeht steigt sie. Das lässt darauf schließen, dass das System in den Aufheiz- und Abkühlphasen nichtlinear ist, was mit Ansatzfunktionen ersten Grades nicht identifiziert werden kann. Dafür eignen sich Ansatzfunktionen zweiten Grades durch deren Nichtlinearität besser. Entscheidend bei diesem System ist aber, dass der Abschnitt zwischen der Aufheiz- und Abkühlphase richtig identifiziert wird, was hier mit Ansatzfunktionen erster Ordnung der Fall ist.

Einzelidentifikation mit dem zweiten Trainingsdatensatz Diese Untersuchung wird nun mit dem zweiten Trainingsdatensatz durchgeführt. Die Ergebnisse sind in den Abbildungen 50 und 51 dargestellt. Es zeigt sich ein umgekehrtes Verhalten. Mit Ansatzfunktionen ersten Grades, ist das Ergebnis deutlich schlechter als mit Ansatzfunktionen zweiten Grades. Die Ansatzfunktionen ersten Grades reichen entgegen der Erwartungen nicht aus, um die logarithmischen bzw. exponentiellen Phasen abzubilden. Erst bei Ansatzfunktionen zweiten Grades schafft SINDYc ein zufriedenstellendes Ergebnis. Dies ist ein Problem bei der simultanen Verwendung beider Trainingsdatensätze, denn es ist fraglich, ob SINDYc es überhaupt schafft das System zu identifizieren.

Während der Tests mit den Trainingsdatensätzen zeigte sich, dass eine Verringerung von r, also eine Vergrößerung der Datenpunktanzahl, eine Verringerung der QA zur Folge hat. Bei $r = 20$ ist es auch möglich, mit Ansatzfunktionen ersten Grades den zweiten Trainingsdatensatz richtig zu identifizieren. Jedoch ergibt ein Vergleich der Koeffizienten, dass diese nicht übereinstimmen.

Verifizierung mit dem Testdatensatz Beim Training beider Trainingsdatensätze und gleichzeitigem Test mit dem vorerst zurückgehaltenem Testdatensatz ergibt sich

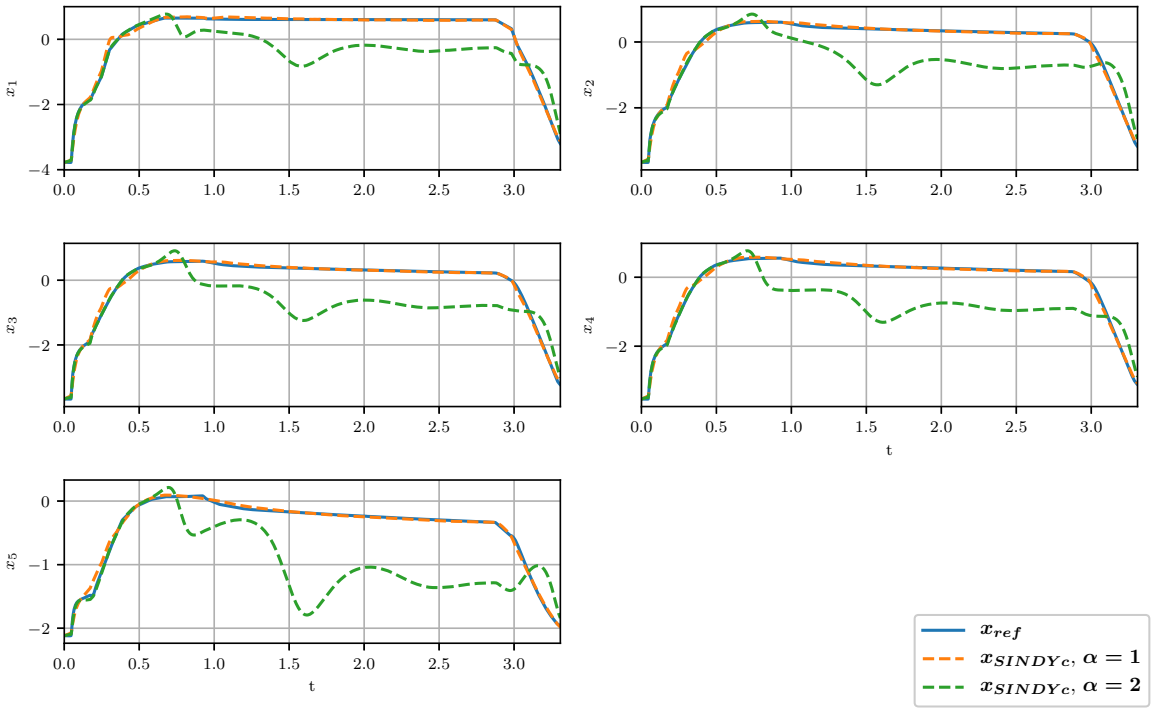

Abbildung 48 – Ergebnis für den Trainingsdatensatz 1 des ersten Prozesses mit SINDYc mit Ansatzfunktionen $\alpha = 1$ und $\alpha = 2$.

das in Abbildung 52 dargestellte Ergebnis für Ansatzfunktionen ersten Grades. Die Abweichung beträgt insgesamt 3843 und pro Datenpunkt 1,2. Aus diesen Werten und der Abbildung 52 ist zu schließen, dass SINDYc es nicht schafft, das System erfolgreich mit Ansatzfunktionen ersten Grades zu identifizieren. Ursache hierfür ist, dass der zweite Trainingsdatensatz unbedingt Ansatzfunktionen zweiten Grades benötigt. Wird das Training mit Ansatzfunktionen zweiten Grades durchgeführt, werden Koeffizienten berechnet, die mit dem Differentialgleichungslöser zu Ergebnissen in der Größenordnung von größer 10^{10} führen. Es wird ein instabiles System identifiziert. Die LASSO-Regression ergibt falsche Koeffizienten. Das kann an Nutzung zweier verschiedener Arbeitspunkte liegen, denn die Identifikation der einzelnen Trainingsdatensätze ergab kein instabiles Verhalten. Eine mögliche Lösung wäre die Verwendung einer anderen Regressionsmethode, mit der stabile Koeffizienten gefunden werden.

Abschließend ist zu den Tests des Prozess 1 mit SINDYc festzustellen:

- die Identifikation mit jeweils nur einem Trainingsdatensatz funktioniert sehr gut, dafür werden jedoch unterschiedliche Polynomgrade für die Ansatzfunktionen benötigt,
- die Identifikation des Gesamtsystems schlägt fehl, weil SINDYc nicht die richtigen Koeffizienten berechnet, die erforderlich wären um den Testdatensatz zu prädizieren.

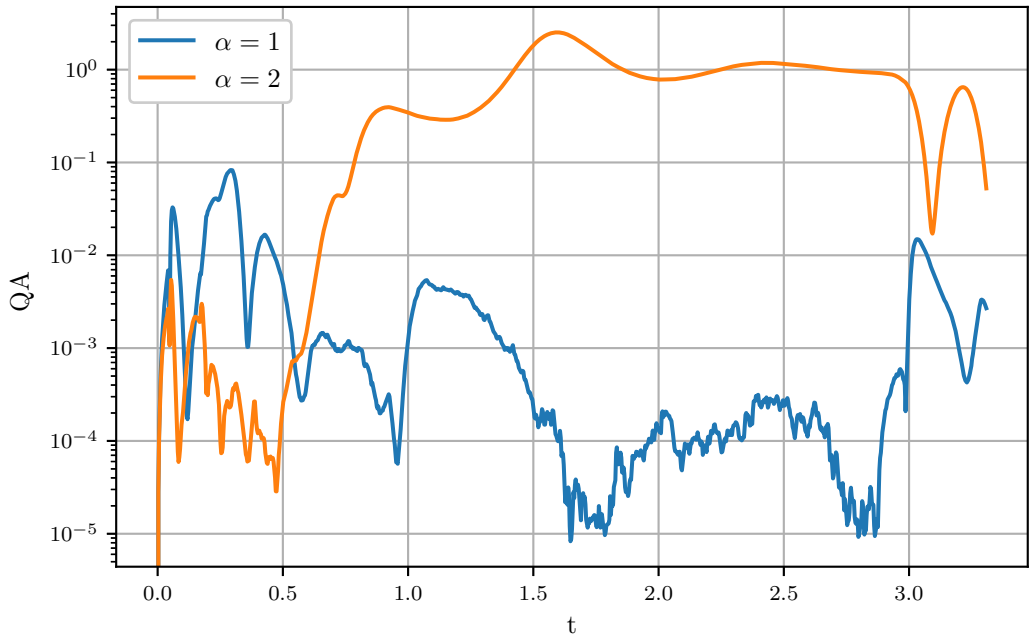

Abbildung 49 – QA für den Trainingsdatensatz 1 des ersten Prozesses mit SINDYc mit Ansatzfunktionen $\alpha = 1$ und $\alpha = 1$.

Ursache hierfür sind die sehr unterschiedlichen Messdaten.

4.2.2 Prozess 2

Training mit dem Trainingsdatensatz Wie bereits beim ersten Prozess, wird zuerst getestet ob das System überhaupt identifiziert werden kann. Dafür wird eine einzelne Messung des Trainingsdatensatzes trainiert und mit der gleichen Messung getestet. Dabei soll festgestellt werden, welcher Grad der Ansatzfunktionen genutzt werden muss. Das Ergebnis für Ansatzfunktionen ersten bis vierten Grades ist in Abbildung 53 dargestellt. Für $\alpha = 1$ und $\alpha = 2$ kommt es zu einem falschen Ergebnis bei x_4. Diese Abweichung ist bei Ansatzfunktionen höheren Grades nicht vorhanden. Außerdem sind die Verläufe an der Stelle der Phasengrenze für $\alpha = 3$ und $\alpha = 4$ besser als für die Ansatzfunktionen geringeren Grades, weil der LASSO-Regression zur Minimierung mehr Parameter zur Verfügung stehen.

Die Schlussfolgerung ist, dass für den Test mit dem Testdatensatz und dem Trainingsdatensatz ein möglichst hoher Ansatzfunktionengrad gewählt werden. Die Herausforderung ist, dass SINDYc das System aus der großen Anzahl an Trainingsmessungen richtig identifiziert. Es ist unklar, wie gut die Nichtlinearität des Systems bei einer großen Trainingsmessungsanzahl identifiziert werden kann. Ob die Identifikation erfolgreich war

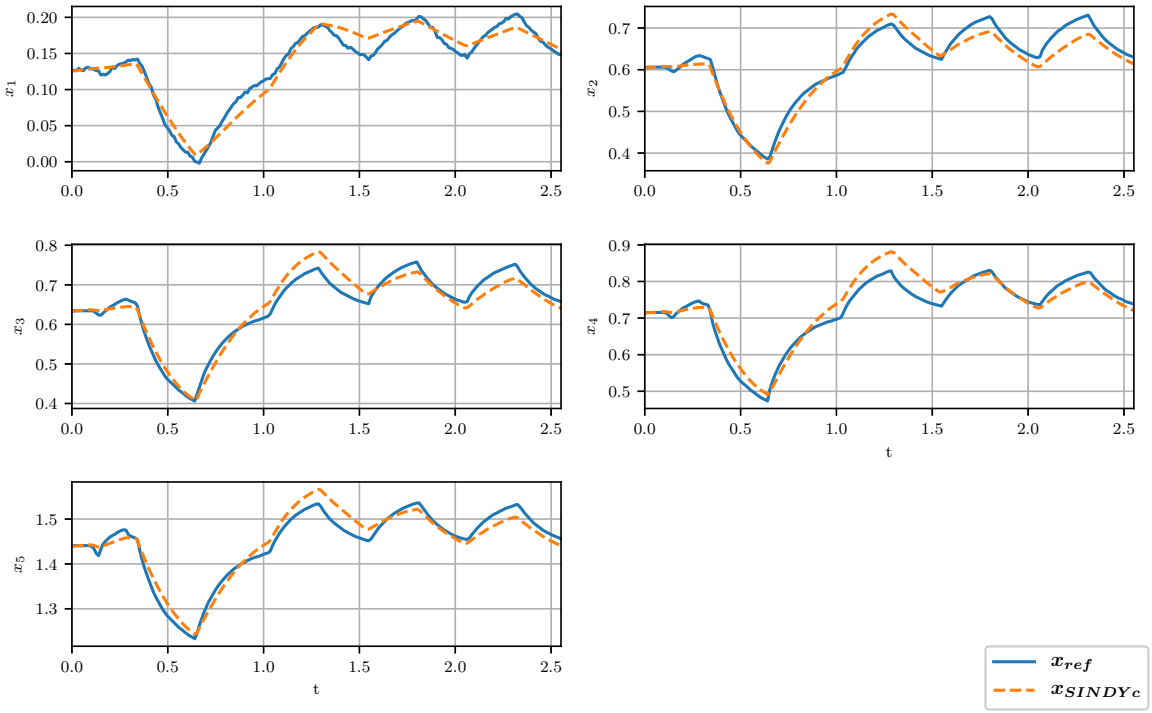

Abbildung 50 – Ergebnis für den Trainingsdatensatz 2 des ersten Prozesses mit SINDYc mit Ansatzfunktionen $\alpha = 1$.

ist daran zu erkennen, wenn bei der Prädiktion die Phasengrenze an der richtigen Stelle berechnet wird. Wichtig ist außerdem, dass der sechste Zustand (die Phasengrenze) unterschiedliche Steigungen bei unterschiedlichen Messungen hat. Demzufolge muss SINDYc auf die richtige Steigung schließen. Außerdem ist zu betrachten, wie sich das Ergebnis mit einer Zunahme der Anzahl der Messungen verändert. Dafür ist entscheidend, wie die QA pro Messung wächst oder fällt.

Verifizierung mit dem Testdatensatz Bei den Tests mit dem Testdatensatz ergaben sich überraschende Ergebnisse. In der Abbildung 54 ist der Test mit den Ansatzfunktionen zweiten Grades dargestellt. Nach dieser Abbildung ist davon auszugehen, dass das System richtig identifiziert wurde. Dem ist aber nicht der Fall, wenn die Gesamtheit der Testmessungen betrachtet wird. Bei vielen Testmessungen kommt es zu erheblichen Abweichungen ab einem bestimmten Zeitpunkt. Diese Abweichung betrifft alle Zustände. Die Annahme, dass mit einer hohen Ansatzfunktionengrad die Genauigkeit steigt, trifft demzufolge nicht zu. Der Grund dafür sind falsche Koeffizienten, die zu einem instabilen System führen.

Die Auswirkungen der falschen Koeffizienten sind auch bei Ansatzfunktionen ersten Grades zu beobachten. Zur Erklärung wird Abbildung 53 genauer betrachtet. Auffällig ist bei Ansatzfunktionen ersten Grades, dass es zu großen Abweichungen beim Zustand

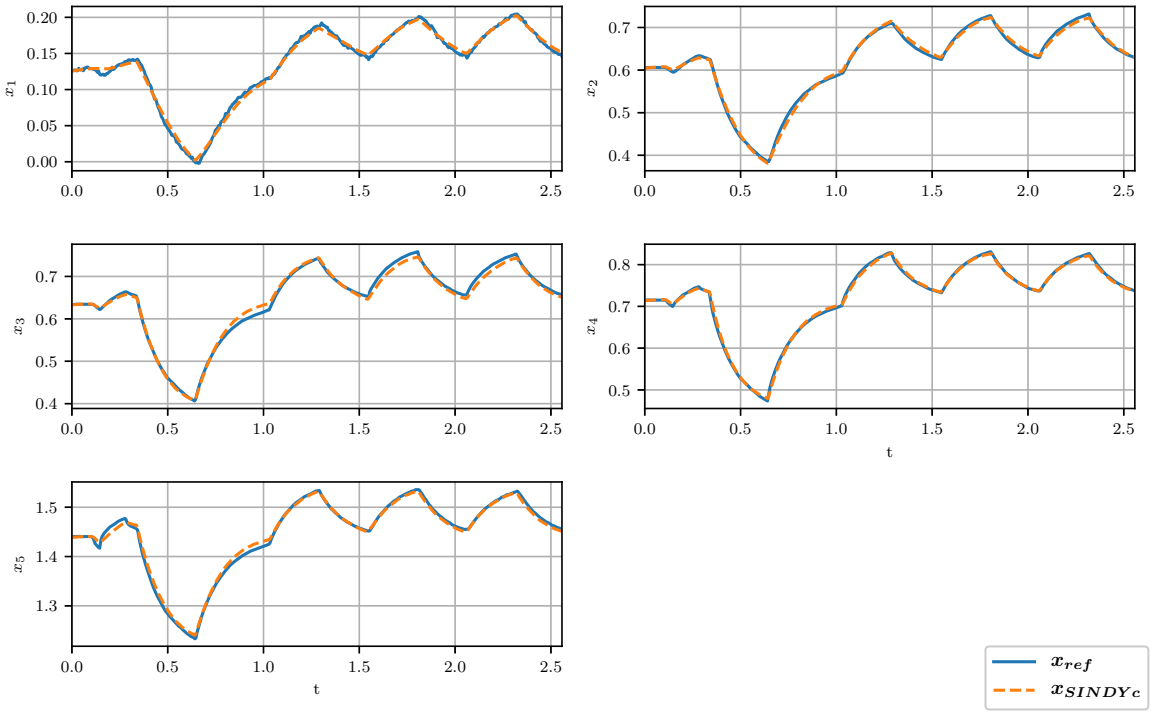

Abbildung 51 – Ergebnis für den Trainingsdatensatz 2 des ersten Prozesses mit SINDYc mit Ansatzfunktionen $\alpha = 2$.

x_4 kommt. Die Trajektorien entfernen sich ab einem bestimmten Punkt exponentiell von der Referenztrajektorie. Bei Ansatzfunktionen zweiten Grades erfolgt eine Verstärkung der Abweichung durch die quadratischen Terme. Demgegenüber stehen stabile Systemidentifikationen mit Ansatzfunktionen dritten bzw. vierten Grades in Abbildung 53.

Beim Test mit dem Testdatensatz sind die identifizierten Systeme mit Ansatzfunktionen dritten bzw. vierten Grades aber auch instabil. Bei Auswertung einzelner ausgewählten Messungen des Testdatensatzes kann aber darauf geschlossen werden, dass das Systemverhalten stabil ist. Ein Muster, bei welchen Messungen die Trajektorien trotz des instabiles Systems richtig prädiziert werden, lässt sich aber nicht finden. Demzufolge muss für die Auswertung immer die Gesamtheit der Testmessungen berücksichtigt werden.

Die Instabilität äußert sich durch die Aufintegration sehr hoher Werte durch den Differentialgleichungslöser. Läuft die Speichervariable über, erfolgt ein Rücksetzer auf einen kleinen Wert. Abschließend wird festgestellt, dass Ansatzfunktionen zweiten Grades nicht das gewünschte Ergebnis liefern, weil das System nicht immer richtig identifiziert wird. Bei Ansatzfunktionen höheren Grades sind die Abweichungen noch wesentlich größer.

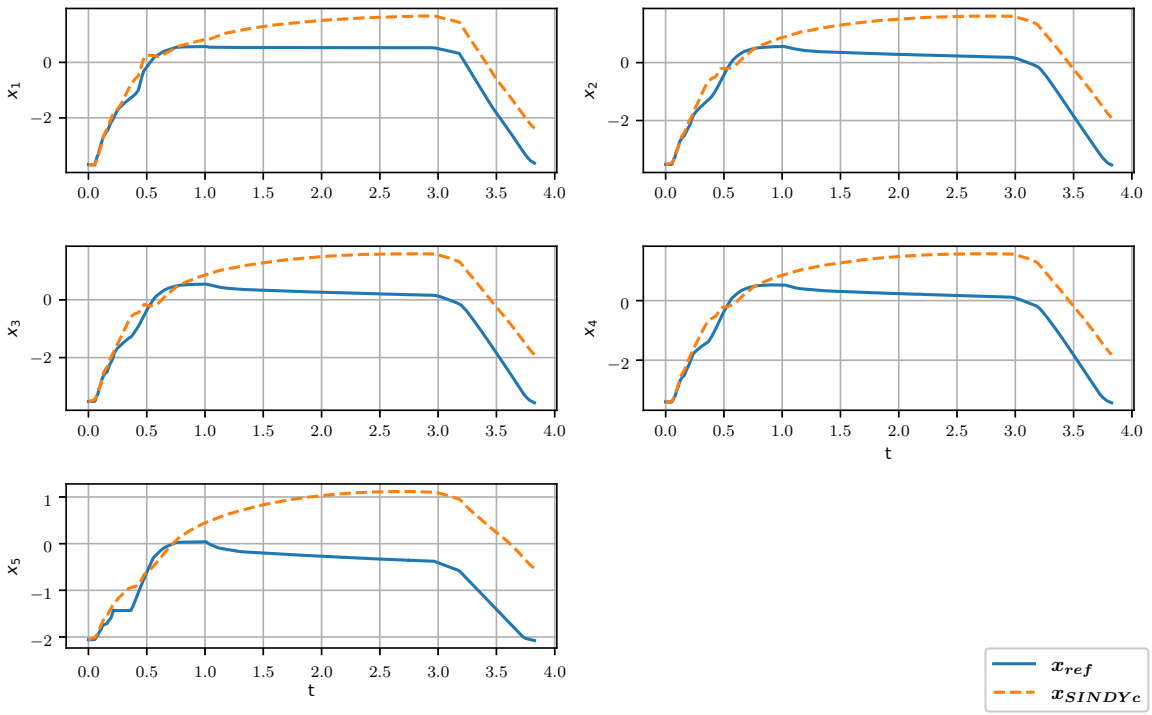

Abbildung 52 – Ergebnis für SINDYc für den ersten Prozess mit beiden Trainingsdatensätzen und dem Testdatensatz mit Ansatzfunktionen $\alpha = 1$.

Demzufolge müssen Ansatzfunktionen erstem Grades verwendet werden, die, wie bereits festgestellt, keine guten Ergebnisse bei einigen Messungen liefern, weil das gefundene System instabil ist, es aber nicht zum Variablenüberlauf kommt. Das Ergebnis ist in Abbildung 55 dargestellt. Allerdings ist die Identifikation an der Stelle der Phasengrenze nicht gut. Diese Stelle ist aber besonders wichtig. Eine mögliche Lösung wäre, nichtlineare Ansatzfunktionen zu nutzen. Damit wäre möglicherweise eine Identifikation nichtlinearer Systeme zu erreichen.

Es ist festzuhalten, dass das System für den letzten Zustand richtig identifiziert wurde und bei allen Testmessungen die prädizierten Trajektorien dieses Zustandes nur geringen Abweichungen zu den Referenztrajektorien aufweisen.

Untersuchung zur Anzahl der Trainingsmessungen Die Entwicklung der Abweichung mit Erhöhung der Anzahl der Trainingsmessungen ist in Abbildung 56 dargestellt. Dabei wurden immer alle Testmessungen prädiziert. Entgegen der Erwartung sinkt die Abweichung nicht für eine größere Anzahl von Trainingsmessungen, sondern verharrt ab 1200 Trainingsmessungen ungefähr auf dem gleichen Niveau und im Bereich zwischen 300 und 1000 Trainingsmessungen kommt es zu einem Anstieg der Abweichung, der sein Maximum bei 600 Trainingsmessungen hat. Der Grund für die großen Abweichungen kann erneut bei der instabilen Identifikation gesucht werden. Es ist möglich, dass für

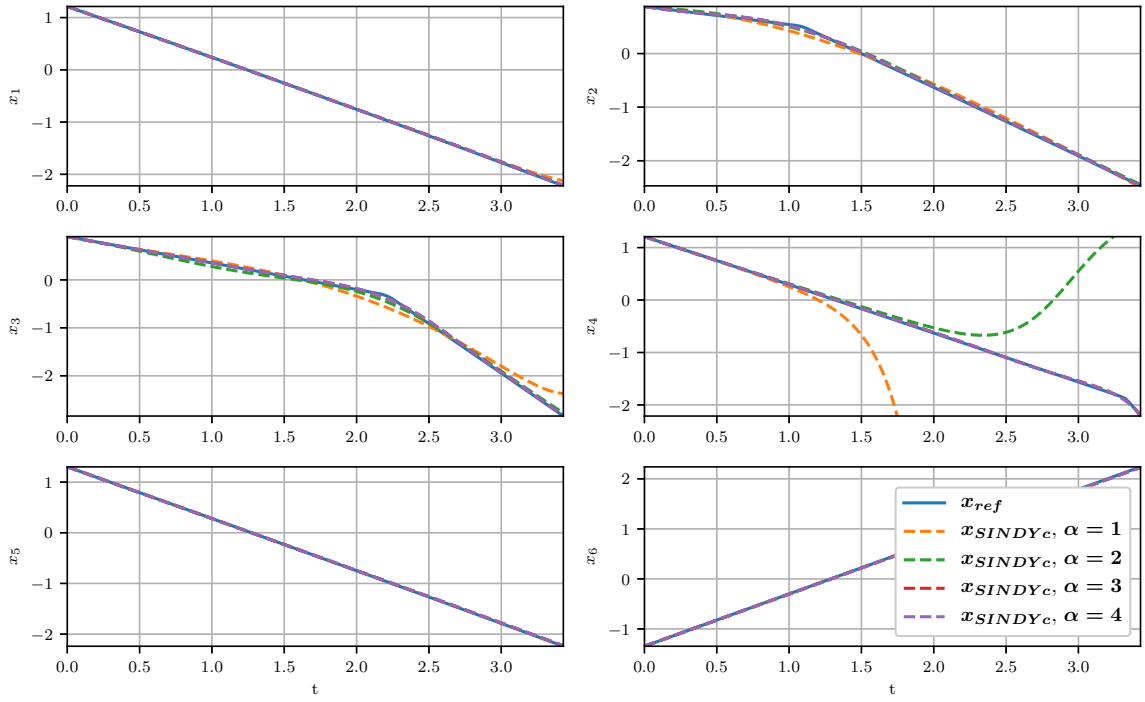

Abbildung 53 – Ergebnis für SINDYc für den zweiten Prozess für zufällig ausgewählte eine Messung für Ansatzfunktionen mit Grad $\alpha = 1, \ldots, 4$

eine Anzahl bis 300 Trainingsmessungen die Identifikation noch stabil ist, mit Zunahme der Trainingsmessungen durch die LASSO-Regression aber keine stabile Koeffizienten mehr gefunden werden können. Erst bei einer sehr großen Anzahl von Trainingsmessungen werden wieder Koeffizienten gefunden, die ein stabiles System ergeben. Eine weitere Erklärung könnte sein, dass eine Teilmenge der Trainingsmessungen nicht zur vollständigen Systemidentifikation ausreicht. Eine vollständige Systemidentifikation erfolgt demzufolge erst nach ca. 1000 Trainingsmessungen. Bis zu einer Anzahl von 300 Trainingsmessungen kommt es nur zu einer sehr beschränkten Systemidentifikation, die aber für die Testmessungen gute Ergebnisse ergibt.

Die Trainingszeiten sind in Tabelle 10 dargestellt. Der Anstieg der Trainingszeit verläuft linear.

4.3 Tests der Prozesse mit dem ODEnet

Für den ODEnet-Ansatz wird ein KNN, wie es in Tabelle 1 dargestellt ist, verwendet. Beim ersten Prozess beträgt die Eingangsdimension 10 und die Ausgangsdimension 5. Beim zweiten Prozess ist die Eingangsdimension 8 und die Ausgangsdimension 6. Die Lernrate ist bei beiden Prozessen 10^{-4}.

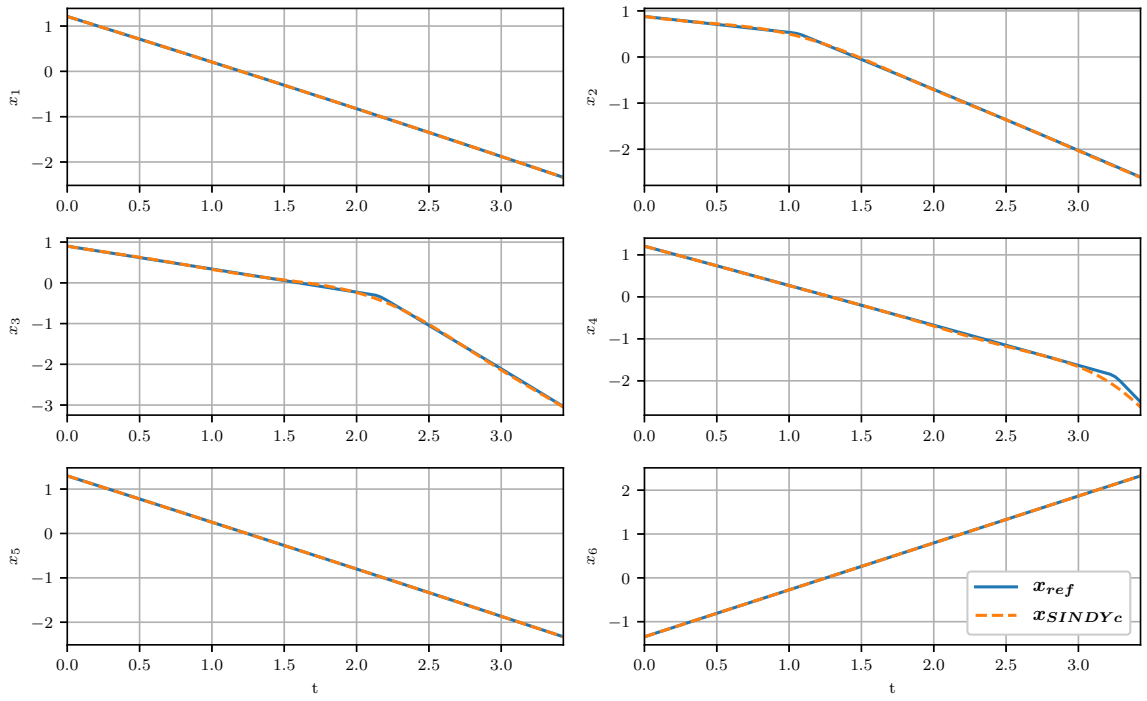

Abbildung 54 – Ergebnis für SINDYc für den zweiten Prozess für eine zufällig ausgewählte Messung des Testdatensatzes für Ansatzfunktionen mit Grad $\alpha = 2$

Trainingsmessungen	Trainingszeit in s
50	4,13
100	16,14
250	99,55
500	233,64
1000	499,80
1500	725,70
1800	896,97

Tabelle 10 – Trainingszeiten bei unterschiedlichen Trainingsmessungsanzahl bei SINDYc

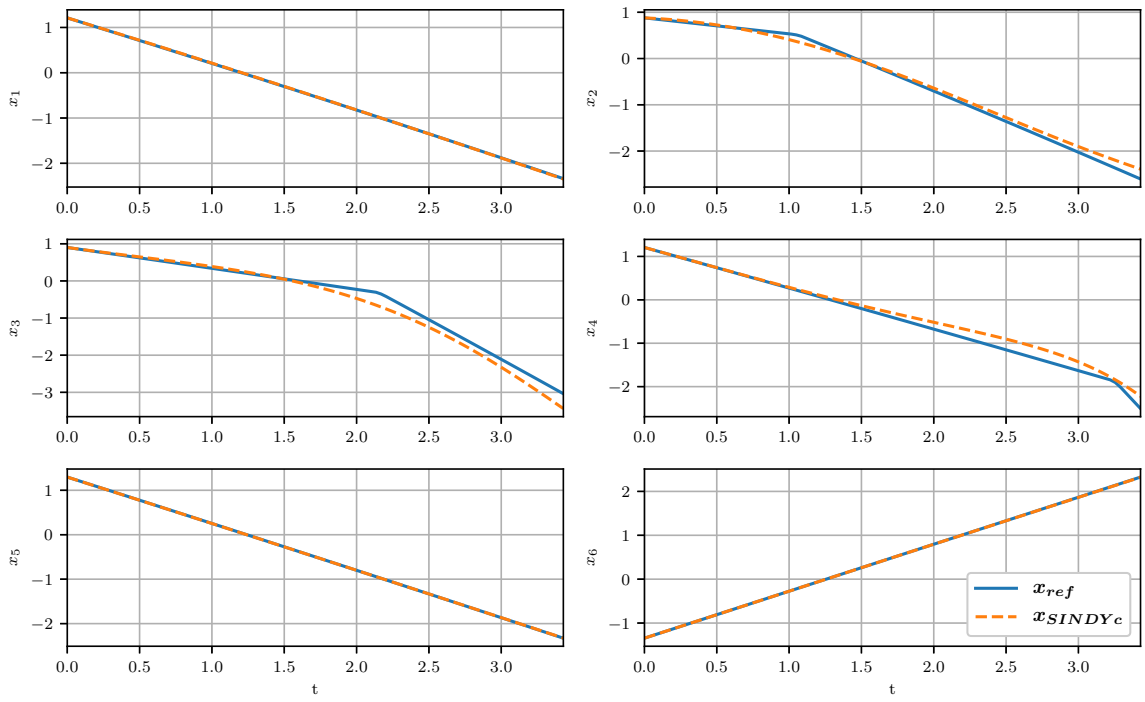

Abbildung 55 – Ergebnis für SINDYc für den zweiten Prozess für eine zufällig ausgewählte Messung des Testdatensatzes für Ansatzfunktionen mit Grad $\alpha = 1$

4.3.1 Prozess 1

Einzelidentifikation der Trainingsdatensätze Entsprechend der Vorgehensweise bei SINDYc, soll auch beim ODEnet zuerst evaluiert werden, ob es die einzelnen Trainingsdatensätze identifizieren kann. Dafür werden die Einstellungen aus den Testsystemen übernommen, um einen guten Ausgangspunkt für etwaige Veränderungen zu haben. Mit den vorher gefundenen Einstellungen schafft es der ODEnet-Ansatz die beiden Trainingsdatensätze zu identifizieren. Die MQA pro Punkt beträgt dabei nur $4 \cdot 10^{-3}$ bzw. $7 \cdot 10^{-4}$. Interessant ist, ob es möglich ist, das System auch aus dem gleichzeitigen Training beider Trainingsdatensätze zu identifizieren. Weil die Einstellungen für die einzelnen Trainingsdatensätze ausreichend waren, werden diese weiter genutzt. Für den Test wird der Testdatensatz genutzt, der nicht für das Training verwendet wird. Nur wenn das ODEnet gut generalisiert, ist eine Identifizierung möglich. Eine Überanpassung ist aufgrund der großen Unterschiede zwischen den beiden Trainingsdatensätzen nahezu nicht möglich.

Verifizierung mit dem Testdatensatz Das Ergebnis für den Testdatensatz zeigt Abbildung 57. Die sehr guten Ergebnisse aus den einzelnen Tests mit den Trainingsdatensätzen können nicht bestätigt werden. Das ODEnet schafft es zwar das System

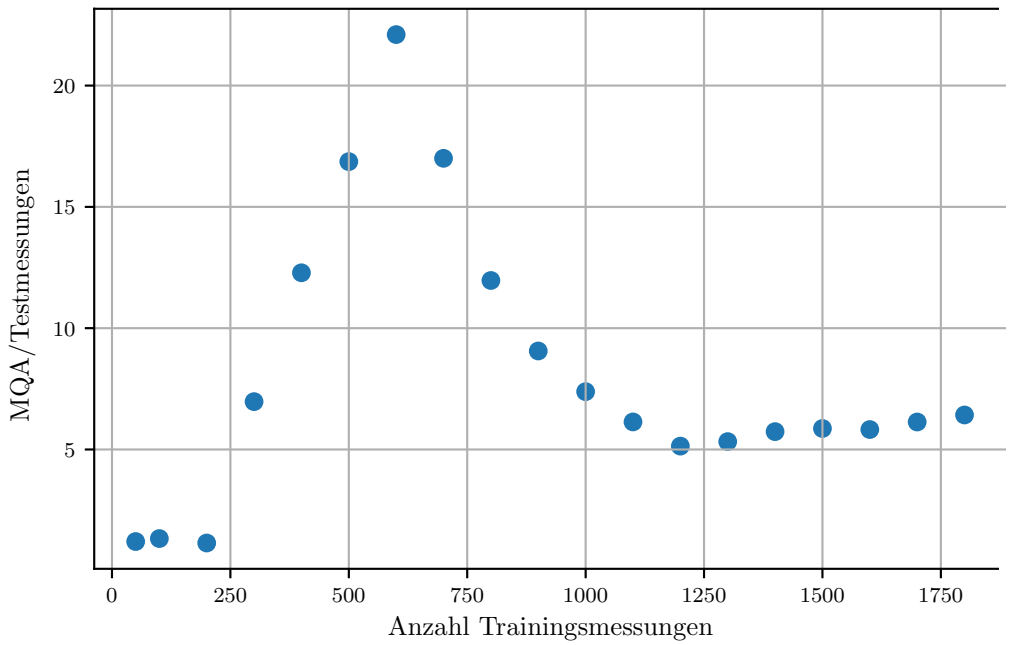

Abbildung 56 – QA pro Testmessung für SINDYc für den zweiten Prozess für eine Testmessung für Ansatzfunktionen mit Grad $\alpha = 1$

ungefähr zu identifizieren, jedoch sind noch große Abweichungen bei den Zuständen zu beobachten. Die MQA pro Punkt liegt bei 0,86, was die Abweichung bestätigt. Das wird auch in Abbildung 58 für alle Zustände verdeutlicht. Ein erheblicher Anstieg der MQA ist an der Stelle $t = 3{,}4$ zu erkennen, weil die Trajektorie sich an dieser Stelle stark verändert. Das ODEnet hat diese Veränderung nicht ausreichend gut gelernt. Es zeigt sich aber auch im Zeitintervall $t = [1{,}25; 3]$ ein Anstieg und ein Absinken der MQA. Das ist genau in dem linearen Bereich der Referenztrajektorie. Der Grund für die Abweichungen sind wahrscheinlich die großen Unterschiede in den Trainingsdatensätzen. Damit ist es dem ODEnet nicht möglich, die Gemeinsamkeiten zu identifizieren und die Gewichte dementsprechend einzustellen. Eine Verbesserung wäre zu erwarten, sobald mehr unterschiedliche Messdaten des gleichen Systems zur Verfügung stünden. In diesem Fall könnten die Gewichte besser angepasst werden. Des Weiteren ist der Testdatensatz sehr ähnlich zu einem der Trainingsdatensätze. Das Training, und damit die Optimierung, erfolgt aber mit beiden Trainingsdatensätzen. Das ODEnet passt die Gewichte so an, dass es bestmögliche Ergebnisse mit beiden Trainingsdatensätzen erzielt. Auffällig ist aber, dass sich die Trajektorien am Ende der Testzeit wieder aneinander annähern. Das spricht dafür, dass das ODEnet das Systemverhalten gelernt hat, aber nicht auf die kleinere Amplitude schließen kann, weil die Trainingsdatensätze eine höhere Amplitude aufwiesen.

Beim ersten Prozess führten eine Erhöhung der Anzahl verdeckter Schichten und eine

Erhöhung der Neuronenanzahl nicht zu signifikanten Verbesserungen. Auffällig war, dass die Lernrate einen erheblichen Einfluss auf das Ergebnis besitzt. Diese musste für gute Ergebnisse auf $5 \cdot 10^{-4}$ gesetzt werden. Wird die Schrittweite zu groß oder klein gewählt, schlägt das Training fehl mit der Folge erheblicher Abweichungen.

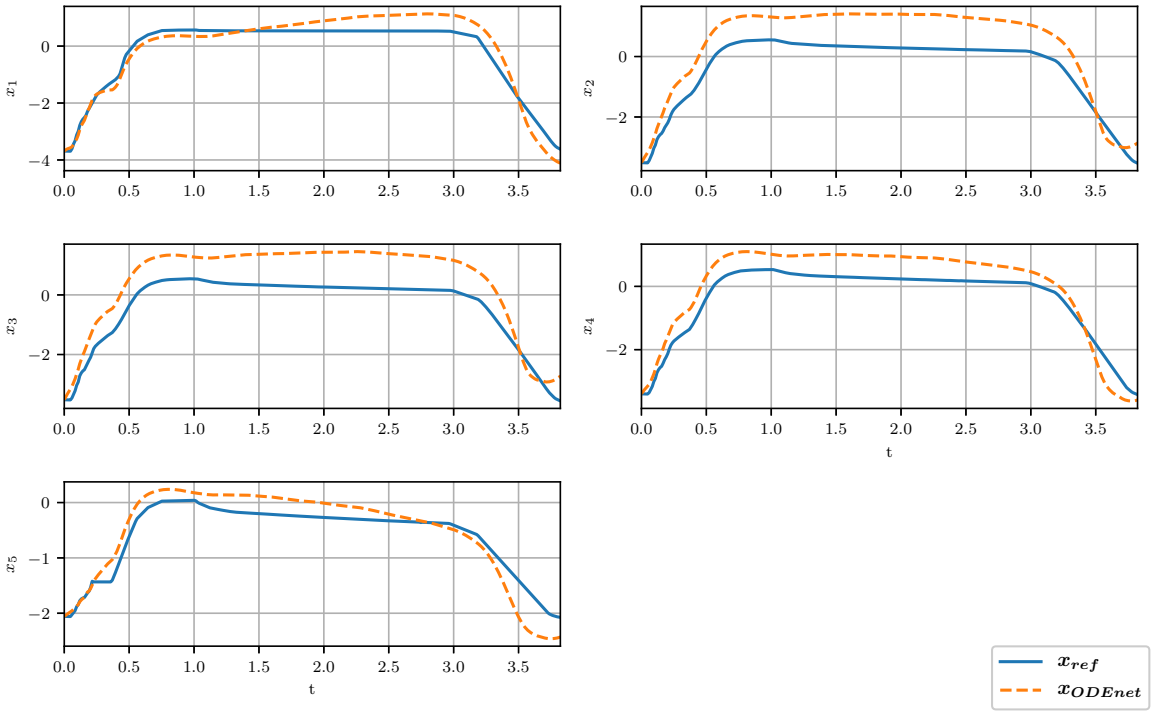

Abbildung 57 – Ergebnis für den Testdatensatz des ersten Prozesses mit dem ODEnet

4.3.2 Prozess 2

Beim zweiten Prozess wurde das ODEnet sofort mit dem Trainingsdatensatz trainiert und dem Testdatensatz getestet, weil Daten ähnlich sind. Damit kann auf die Einzelidentifikation des Trainingsdatensatzes verzichtet werden.

Verifizierung mit dem Testdatensatz Das Ergebnis für den zweiten Prozesses ist in der Qualität ähnlich zu dem des ersten Prozesses. Wie in Abbildung 59 dargestellt, kommt es bei allen Zuständen zu Abweichungen. Die MQA liegt pro Punkt bei 0,34. Allerdings schafft es das ODEnet das nichtlineare Verhalten abzubilden, wenn auch nicht mit einer sehr guten Genauigkeit. Im Vergleich zu SINDYc ist die Identifikation des sechsten Zustandes jedoch wesentlich schlechter. Dafür kommt es beim ODEnet nicht zu Instabilitäten, wie es bei SINDYc der Fall war.

Bei den Abweichungen des ODEnets ist auffällig, dass das identifizierte System immer

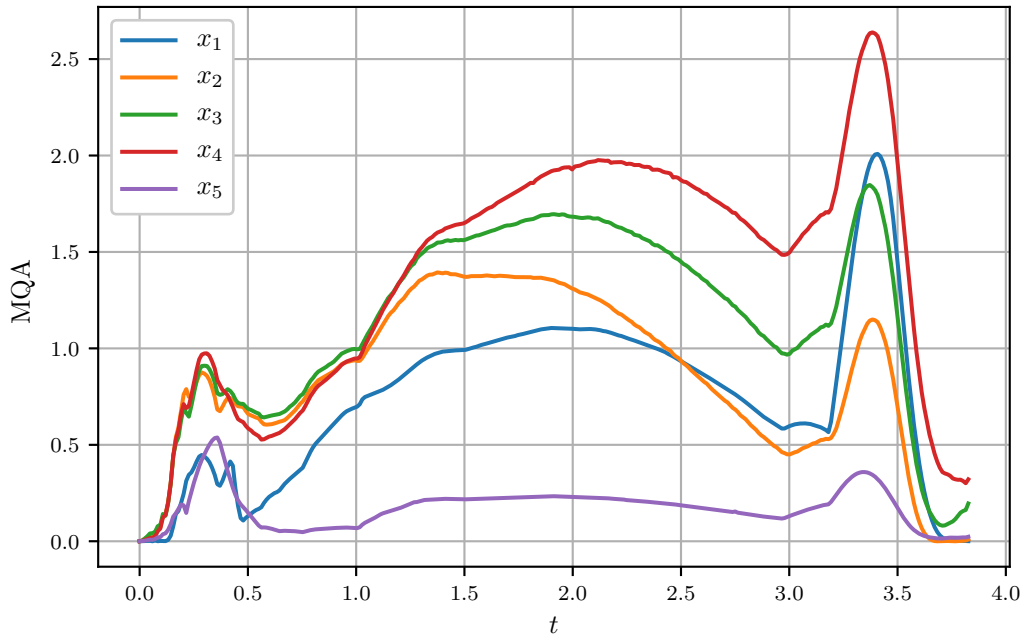

Abbildung 58 – MQA für den Testdatensatz des ersten Prozesses mit dem ODEnet

unterhalb des Referenzsystems liegt. Dieser Umstand ist den Trainingsdaten geschuldet, denn die Phasengrenze bewegt sich in den Trainingsmessungen immer von kleinen zu großen t. Da das ODEnet eine Generalisierung durchführt, werden die Gewichte so angepasst, dass ein Optimum für alle Trainingsmessungen gefunden wird. Dieses Optimum liegt unterhalb dieser Referenztrajektorie, weswegen es zu dieser Abweichung kommt. Eine Ausnahme bildet hier der sechste Zustand, bei dem sich sich nur der Anstieg ändert. Hier kann das ODEnet aus den Eingangsdaten den Zustand nicht genau prädizieren. Der Grund dafür kann in der sehr großen Ähnlichkeit der Eingangstrajektorien liegen, die sich nur im Anstieg unterscheiden. SINDYc belegt die Eingänge mit einem Koeffizienten, während das ODEnet die Gewichte während des Trainings anpassen muss. Sind die Eingangstrajektorien sehr ähnlich, ist es für das ODEnet schwer, eine Generalisierung der Unterschiede zwischen den Eingangstrajektorien zu finden, weswegen falsche Gewichte berechnet werden.

Des Weiteren fällt auf, dass die Abweichungen erst nach einer bestimmten Zeit einsetzen. Das spricht dafür, dass die Gewichte überwiegend richtig optimiert wurden, aber das System nicht vollständig generalisiert wurde. Dieses Problem konnte auch nach einer Erhöhung der Neuronenanzahl oder der Anzahl verdeckter Schichten nicht gelöst werden.

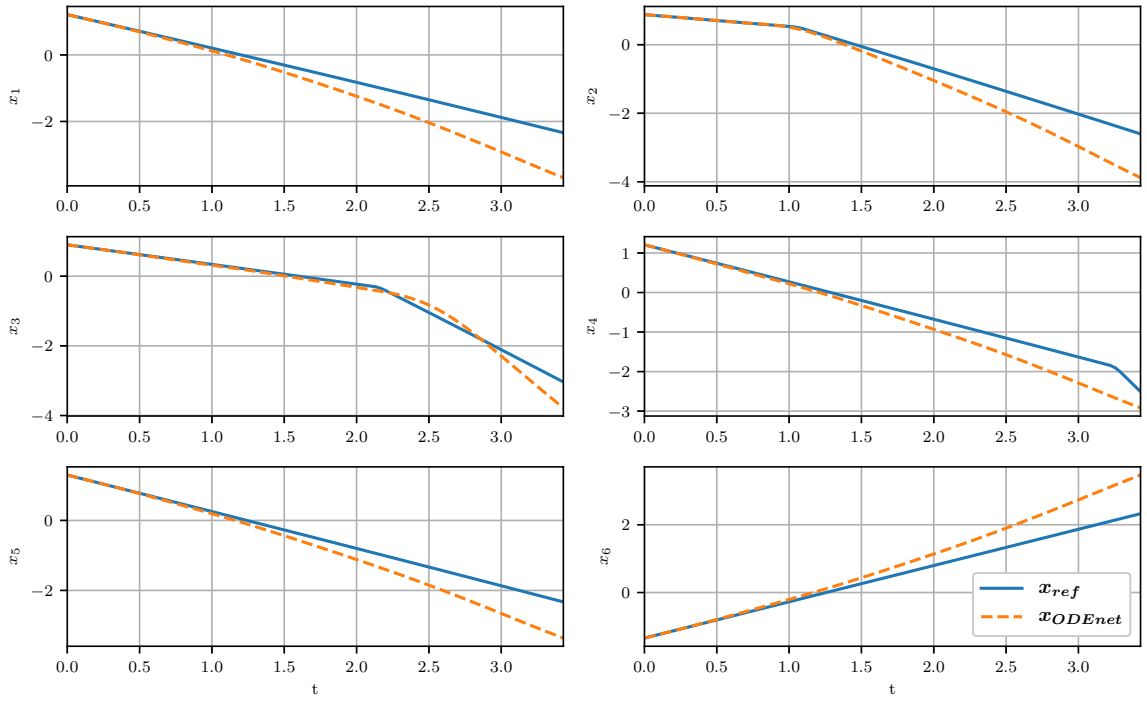

Abbildung 59 – Ergebnis für den Testdatensatz des zweiten Prozesses mit dem ODEnet

4.4 Kombination von SINDYc und ODEnet

Für die Kombination der beiden Systemidentifikationsansätze sind zwei Varianten vorstellbar. Die erste Variante basiert auf dem Fehler der bei der Prädiktion des Systems von SINDYc begangen wird, während der zweite auf der Aufteilung der Dynamik in zwei Teildynamiken aufbaut. Die Kombination wird nur für den zweiten Prozess durchgeführt.

Variante 1 Eine Verbesserung der Ergebnisse könnte die Kombination beider Systemidentifikationsansätze ergeben. Eine mögliche Idee ist, dass SINDYc das System zuerst identifiziert und die Abweichungen die zu den Referenzdaten vorhanden sind, vom ODEnet gelernt werden sollen. Dadurch wird ein System des Fehlers erstellt und das ODEnet könnte dieses System identifizieren. Das verwendete ODEnet ist das gleiche, wie es auch bei den vorangegangenen Identifizierungen benutzt wurde.

Für das Training sollen die Eingänge für das ODEnet die Referenzeingänge der jeweiligen Testmessungen bleiben. Die Zuständen berechnen sich aus der Differenz der Referenztrajektorie zu der prädizierten Trajektorie von SINDYc zu

$$\Delta \boldsymbol{x} = \boldsymbol{x}_{\text{ref}} - \boldsymbol{x}_{\text{SINDYc}}. \tag{4.2}$$

Nach dem Training beträgt die MQA pro Punkt der Abweichung bei 0,026. Dabei

muss jedoch beachtet werden, dass die $\Delta \boldsymbol{x}$ maximal $3 \cdot 10^{-1}$ groß ist. Auch bei der Betrachtung der identifizierten Trajektorien in Abbildung 60 ist zu erkennen, dass $\Delta \boldsymbol{x}$ nicht gut identifiziert wird. Dass dem so ist liegt zum einen daran, dass es keinen sinnvollen Zusammenhang zwischen den Eingängen und $\Delta \boldsymbol{x}$ vorhanden ist. Zu anderen ist Anzunehmen, dass $\Delta \boldsymbol{x}$ für die verschiedenen Testmessungen sehr unterschiedlich ist. Durch diese zwei Gegebenheiten ist es dem ODEnet nicht möglich, einen Zusammenhang zur Generalisierung herzustellen, weswegen diese Kombination der beiden Ansätze fehlschlägt.

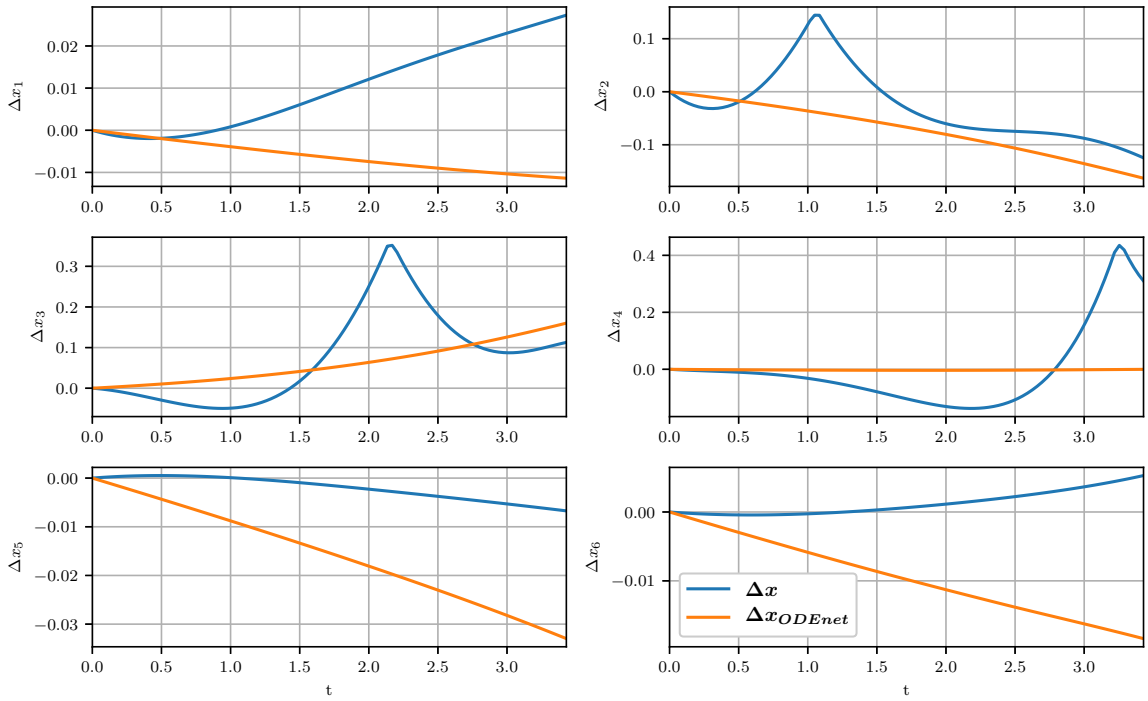

Abbildung 60 – Ergebnis für $\Delta \boldsymbol{x}$ den Testdatensatz des zweiten Prozesses mit dem ODEnet

Variante 2 Eine weiter mögliche Kombination der beiden Ansätze besteht darin, dass angenommen wird, dass die zu identifizierende Dynamik in zwei Dynamiken unterteilt wird. Ausgegangen wird dabei von (2.1):

$$\begin{aligned} \dot{\boldsymbol{x}}(t) &= f(\boldsymbol{x}(t), \boldsymbol{u}(t)) \\ &= \underbrace{f_1(\boldsymbol{x}(t), \boldsymbol{u}(t))}_{\text{SINDYc}} + \underbrace{f_2(\boldsymbol{x}(t), \boldsymbol{u}(t))}_{\text{ODEnet}} \end{aligned} \tag{4.3}$$

Bei dieser Variante wird zuerst $f_1(\boldsymbol{x}(t), \boldsymbol{u}(t))$ von SINDYc berechnet. Daraufhin erfolgt das Trainieren des ODEnet und das Lösen der Differentialgleichung. Die Gewichte im ODEnet werden vom Optimierer angepasst. Die Dynamik $f_2(\boldsymbol{x}(t), \boldsymbol{u}(t))$ ist im ODEnet

enthalten und kann während des Trainingsprozess durch die Anpassung der Gewichte verändert werden, während die Funktion $f_1(\boldsymbol{x}(t), \boldsymbol{u}(t))$ nach dem Training von SINDYc feststeht. Damit ist eine ungefähre Identifikation der Systemdynamik durch SINDYc möglich und das ODEnet muss nur die Abweichung von der Referenzdynamik lernen.

Dieser Ansatz wurde in der vorliegenden Arbeit nicht implementiert und getestet.

4.5 Vergleich der Ergebnisse

Für den Vergleich der beiden Ansätze werden die MQA und die Trainingszeit für das Training und den Test mit den Prozessen verglichen. Dieser Vergleich ist in Tabelle 11 dargestellt. Außerdem erfolgt ein qualitativer Vergleich der Ergebnisse. Der Vergleich stützt sich ausschließlich auf die Erkenntnisse, die aus den Tests mit den Prozessen gewonnen wurden.

Beim ersten Prozess erzielt das ODEnet bessere Ergebnisse, wie anhand der MQA von SINDYc und ODEnet in Tabelle 11 zu erkennen ist. Der Unterschied ist zwar nur gering, jedoch wurde beim Test mit SINDYc festgestellt, dass vor allem am Ende die Abweichungen erheblich steigen, weil die Koeffizienten nicht exakt berechnet wurden. Dieses Verhalten hat sich auch schon bei den Testsystemen gezeigt. Dem gegenüber steht aber die erheblich längere Trainingszeit vom ODEnet. Das ODEnet kann bei diesem Prozess die Schwäche von SINDYc ausnutzen. SINDYc benötigt bei sehr unterschiedlichen Trajektorien mehr Trainingsdaten, um das System zu identifizieren. Das ODEnet kann durch die Generalisierung einen besseren Zusammenhang zwischen Eingangsdaten und Zuständen herstellen.

Der Vergleich des zweiten Prozesses muss zweigeteilt werden. Für den Fall, dass SINDYc ein stabiles System identifiziert ist das Identifizierungsergebnis deutlich besser als beim ODEnet. Das ist aber nur bei Ansatzfunktionen ersten Grades der Fall und auch bei diesen kann es bei SINDYc zu erheblichen Abweichungen kommen. SINDYc benötigt für die Ergebnisse nur 1/7 der Trainingszeit des ODEnet. Hier ist SINDYc zu bevorzugen. Auch bei nochmaliger Betrachtung von Abbildung 55 ist ein sehr gutes Ergebnis von SINDYc zu erkennen. Festzuhalten ist aber, dass die Phasengrenze vom ODEnet besser identifiziert wird und die großen Abweichungen dadurch entstehen, dass für große Zeiten t die Prädiktion nicht genau ist.

Ein großes Problem im Zusammenhang mit dem zweiten Prozess und SINDYc ist aber, das instabile Systeme identifiziert werden, sobald der Ansatzfunktionengrad erhöht wird. In diesem Fall kommt es zu sehr großen Abweichungen und das ODEnet ist zu bevorzugen. Nichtlineare Ansatzfunktionen könnten möglicherweise bei SINDYc wesentlich bessere Ergebnisse erzielen, insofern ein stabiles System identifiziert wird.

Bei beiden Prozessen und Systemidentifikationsansätzen zeigte sich über die Zeit ein

Prozess	Ansatz	MQA pro Testdatensatz	Trainingszeit in s
1	SINDYc	315,46	14,10
1	ODEnet	276,17	611,00
2	SINDYc	1,21	896,97
2	ODEnet	52,94	6480,00

Tabelle 11 – Vergleich der MQA und der Trainingszeit von SINDYc und ODEnet anhand der Testdatensätze

Anstieg der QA. Dass die Abweichungen über die Zeit zunehmen, zeigt sich sehr deutlich in Abbildung 61. Hier wurden alle Prädiktionen von SINDYc und ODEnet der Testmessungen des zweiten Prozesses verglichen. Dabei stellt die durchgezogene Linie jeweils den Mittelwert der Abweichung und die transparenten Bereiche das 95%-ige Konfidenzintervall dar. Anhand dieser Abbildung zeigt sich, dass das ODEnet eine wesentlich höhere QA hat.

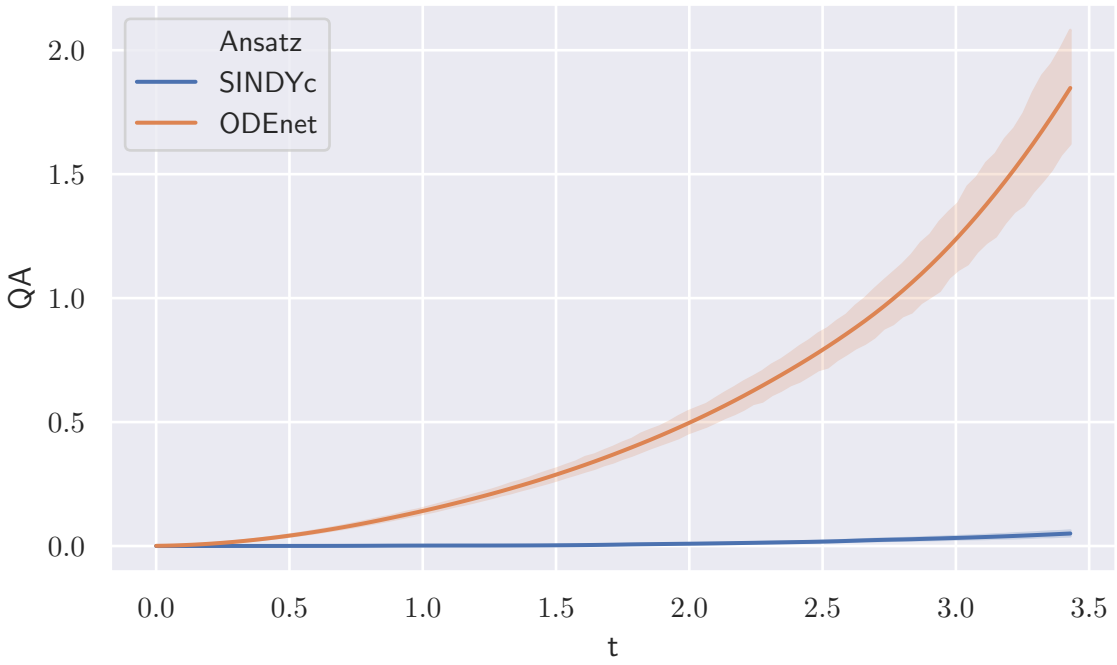

Abbildung 61 – Vergleich der QA für alle Testmessungen für ODEnet und SINDYc des zweiten Prozesses

Kapitel 5

Zusammenfassung und Ausblick

5.1 Zusammenfassung

Um ein gutes Züchtungsergebnis im VGF-Kristallzüchtungsprozess zu erhalten, sind die Temperatur der Schmelze und der Temperaturgradient an der Phasengrenze auf bestimmten Werten zu halten. Dafür ist eine Regelung der Heizer an den Stirn- und Mantelseiten mit einer MPC erforderlich. Das für die Regelung nötige Systemmodell ist unbekannt und sollte in dieser Arbeit gefunden werden.

Zum Finden des Systemmodells wurde zuerst eine Literaturrecherche durchgeführt, nach deren Ergebnis die Ansätze SINDYc und ODEnet sich als besonders zielführend erwiesen. Neben diesen beiden Ansätzen gibt es, insbesondere auf dem Gebiet der KNN, noch weitere vielversprechende Ansätze, wie die LSTM-KNN. Auch die komprimierte Erfassung wurde in einigen Veröffentlichungen zur Systemidentifikation genutzt.

Der SINDYc-Ansatz beruht auf der Nutzung von Ansatzfunktionen und der Optimierung. Die Ansatzfunktionen werden für bekannte Messdaten und Eingangsdaten berechnet und dann mit Hilfe der LASSO-Regression optimiert. Dabei entsteht eine dünn besetzte Matrix, die die Koeffizienten für die Ansatzfunktionen der jeweiligen Zustände enthält. Damit ist das System identifiziert und kann im modellprädiktiven Regler verwendet werden.

Der ODEnet-Ansatz basiert auf der Nutzung KNN und eines Differentialgleichungslösers. Dabei bildet ein KNN die Dynamik der DGL und der Differentialgleichungslöser löst diese. Dabei erfolgt durch einen Optimierer auf Basis eines Kostenfunktionals die Optimierung der Gewichte und Biase des KNN.

Die beiden Ansätze wurden zuerst mittels dreier verschiedener Testsysteme getestet. Die Testsysteme wurden mit Eingängen berechnet, damit ein möglichst regelungstechnisch reales Ergebnis erzielt werden kann. SINDYc schaffte es, die Systeme mit Eingang sehr gut zu identifizieren. Auch bei einer Rauschüberlagerung lieferte SINDYc bis zu einem bestimmten Grad noch gute Ergebnisse. Allerdings kann es durch geringe

Koeffizientenabweichungen zur Abweichungen vom Referenzsystem kommen, sobald die Systembetrachtung über eine langen Zeitraum erfolgt.

Das ODEnet erzielte bei den identischen Testsystemen schlechtere Ergebnisse. Insbesondere bei unterschiedlichen Eingängen waren keine guten Identifikationsergebnisse erzielbar. Auch bei einer Rauschüberlagerung der Trainingsdaten konnte das ODEnet die guten Ergebnisse von SINDYc nicht erreichen. Des Weiteren war die Trainingszeit für das ODEnet wesentlich länger als für SINDYc, was an der Optimierung der vielen Freiheitsgrade des KNN liegt.

Nach diesen Ergebnissen wurde der Einfluss einiger Parameter auf die Ergebnisse untersucht. Dabei stellte sich heraus, dass bei SINDYc insbesondere der 1-Norm-Penalisierungsparameter λ und der Potenzfaktor D eine großen Einfluss haben, weil diese Parameter die dünne Besetztheit der Koeffizientenmatrix beeinflussen. Auch der Grad der Ansatzfunktionen ist für ein gutes Identifikationsergebnis bedeutend. Wird er zu groß gewählt, sind zu viele Freiheitsgrade vorhanden, wird er zu klein gewählt kann das System nicht identifiziert werden.

Beim ODEnet ist für einen Trainingserfolg insbesondere die Neuronenanzahl und die Anzahl der verdeckten Schichten entscheidend. Es wird eine Mindestanzahl an Neuronen und Anzahl verdeckter Schichten benötigt. Eine weitere Steigerung beider Parameter führte aber nicht zu einem verbessertem Ergebnis, sondern nur zu einer erheblichen Verlängerung der Trainingszeit.

Zuletzt wurden reale Anlagendaten für die Tests beider Ansätze verwendet. Dabei stellte sich heraus, dass die Ergebnisse beider Ansätze höhere Abweichungen als bei den Testsystemen aufweisen. Die Koeffizienten werden vom SINDYc-Ansatz beim ersten Prozess nicht richtig gefunden. Beim zweiten Prozess kommt es zu guten Ergebnissen bei Ansatzfunktionen ersten Grades, jedoch bei Ansatzfunktionen höheren Grades bei einigen Testmessungen zu sehr hohen Abweichungen, weil ein instabiles System identifiziert wird.

Das ODEnet schafft es das erste System ungefähr zu identifizieren, allerdings sind die Abweichung erheblich. Der Grund dafür ist die unzureichende Menge an unterschiedlichen Trainingsdaten. Beim zweiten Prozess erreicht das ODEnet gute Ergebnisse, wobei es aber im zeitlichen Verlauf zu höheren Abweichungen kommt. Die Phasengrenze wird aber gut identifiziert und es führt zu einem stabilen Ergebnis.

Eine erste Kombination der beiden Ansätze führte nicht zum Erfolg.

Zusammenfassend ist festzuhalten, dass SINDYc und das ODEnet sehr von den Trainingsdaten abhängig sind. Sind Trainingsdaten in ausreichender Zahl vorhanden und können beide Ansätze gut generalisieren, sind sie zur Systemidentifikation von Nutzen. Stehen aber keine ausreichenden Trainingsdaten zur Verfügung oder sind die Trainingsdaten nicht ausreichend diversifiziert, schlagen beide Ansätze fehl. Es kommt zu großen Abweichungen.

5.2 Ausblick

Die Tests mit beiden Ansätzen weisen auf Gesichtspunkte hin, die in nachfolgenden Arbeiten einer genaueren Betrachtung unterzogen werden sollten:

- Zielführend für eine gute Systemidentifikation könnte eine Kombination der beiden Ansätze sein. Das ODEnet kann nichtlineare Systeme besser identifizieren. ODEnet sollte deswegen für den nichtlinearen Teil der Systemdynamik verwendet werden, während SINDYc den linearen Teil identifiziert. Eine mögliche Kombination der beiden Ansätze wurde in Abschnitt 4.4 vorgestellt und sollte implementiert werden.

- Das in dieser Arbeit implementierte KNN besteht aus dichten Schichten. In der Literatur wird zur Systemidentifikation in einigen Fällen ein LSTM-KNN verwendet. Das ist eine Art eines RNN und hat den Vorteil, dass es intern Daten merken kann. Das ist insbesondere für zeitliche Verläufe sinnvoll, wie sie hier vorliegen. Es wurden anfängliche Tests mit einem LSTM-KNN durchgeführt, die aber keine guten Ergebnisse erbracht haben. Es ist aber anzunehmen, dass die LSTM-KNN in Verbindung mit dem ODEnet-Ansatz, eine erhebliche Reduktion der Neuronenanzahl und der Anzahl verdeckter Schichten und damit der Trainingszeit zur Folge haben.

- Ein wichtiger und untersuchungswürdiger Aspekt ist die Stabilität von SINDYc. Beim zweiten Prozessdatensatz treten bei Ansatzfunktionen ab dem zweiten Grad Abweichungen in der Größenordnung mehrerer Zehnerpotenzen auf. Hier kann es möglicherweise helfen, eine andere Regressionsmethode, wie zum Beispiel die *Ridge-Regression*, zu nutzen. Damit verändert sich auch die dünne Besetztheit der Koeffizientenmatrix $\boldsymbol{\Xi}$. Ein weiterer zu untersuchender Aspekt ist das Finden eines optimalen λ für die Optimierung. Das Optimierungsproblem müsste umformuliert werden und mittels des Pareto-Optimums der optimale Zustand zwischen dünner Besetztheit und Prädiktionsgenauigkeit gefunden werden.

- Eine weitere Verbesserung des Identifikationsergebnisses kann die Beachtung weiterer Ansatzfunktionen bei SINDYc ergeben. Derzeit sind nur Polynome als Ansatzfunktionen implementiert. Insbesondere die Nichtlinearitäten könnten durch die Nutzung nichtlinearer Funktionen besser identifiziert werden. Die Implementierungsmöglichkeiten dafür sind bereits vorhanden.

- Interessant ist auch, ob der Ansatz der komprimierten Erfassung mit der SINDYc-Implementierung funktioniert. Erste Versuche wurden in [28] und [20] beschrieben.

- Als letzter Schritt ist der Test der identifizierten Systeme mit einer MPC erforderlich, denn nur so kann eine finale Aussage über die Güte der Systeme getroffen werden.

Literatur

[1] Ament, C.: *Modellbildung, Identifikation und Simulation dynamischer Systeme*. Skript. Universität Augsburg, Fakultät für Angewandte Informatik. URL: https://www.informatik.uni-augsburg.de/de/lehrstuehle/rt/lehre/Skript/Skript-Modellb-Ident-und-Sim-dyn-Systeme-14.pdf.

[2] Bronstein, I. N. u. a.: *Taschenbuch der Mathematik (Bronstein)* -. 10. Aufl. Haan-Gruiten: Europa Lehrmittel Verlag, 2016.

[3] Brunton, S. L., Proctor, J. L. und Kutz, J. N.: "Discovering governing equations from data by sparse identification of nonlinear dynamical systems". In: *Proceedings of the National Academy of Sciences* 113.15 (2016), S. 3932–3937. DOI: 10.1073/pnas.1517384113.

[4] Brunton, S. L., Proctor, J. L. und Kutz, J. N.: "Sparse Identification of Nonlinear Dynamics with Control (SINDYc)". In: *IFAC-PapersOnLine* 49.18 (2016). 10th IFAC Symposium on Nonlinear Control Systems NOLCOS 2016, S. 710 –715. DOI: https://doi.org/10.1016/j.ifacol.2016.10.249.

[5] Brunton, S. L., Proctor, J. L. und Kutz, J. N.: "Supporting Information for: Discovering governing equations from data". In: *IFAC-PapersOnLine* (2016).

[6] CGS, P.: *Vertical Gradient Freeze (VGF)*. Online; Stand 14. November 2019. 2019. URL: https://www.pvatepla-cgs.com/technologien/vertical-gradient-freeze-vgf/.

[7] Chen, R. T. Q. u. a.: *Neural Ordinary Differential Equations*. 2018. arXiv: 1806.07366 [cs.LG].

[8] Dittmar, R. und Pfeiffer, B.-M.: *Modellbasierte prädiktive Regelung - Eine Einführung für Ingenieure*. Berlin: Walter de Gruyter, 2009.

[9] Glorot, X., Bordes, A. und Bengio, Y.: "Deep Sparse Rectifier Neural Networks". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Hrsg. von Gordon, G., Dunson, D. und Dudík, M. Bd. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, 2011, S. 315–323. URL: http://proceedings.mlr.press/v15/glorot11a.html.

[10] Gonzalez, J. und Yu, W.: "Non-linear system modeling using LSTM neural networks". In: Bd. 51. Juni 2018, S. 485–489. DOI: 10.1016/j.ifacol.2018.07.326.

[11] Hochreiter, S. und Schmidhuber, J.: "Long Short-term Memory". In: *Neural computation* 9 (Dez. 1997), S. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[12] Hofmann, S.: "Identifikation von nichtlinearen mechatronischen Systemen auf der Basis von Volterra–Reihen". 203.

[13] Isermann, R.: *Identifikation dynamischer Systeme 1 - Grundlegende Methoden*. Berlin Heidelberg New York: Springer-Verlag, 2013.

[14] Langheinrich, D.: "Das Vertical Gradient Freeze-Verfahren ohne Tiegelkontakt". 2012. URL: http://slubdd.de/katalog?TN_libero_mab2.

[15] Lau, S.: *Learning Rate Schedules and Adaptive Learning Rate Methods for Deep Learning*. Online; Stand 31. Januar 2020. 2017. URL: https://towardsdatascience.com/learning-rate-schedules-and-adaptive-learning-rate-methods-for-deep-learning-2c8f433990d1.

[16] Ljung, L.: *System Identification - Theory for the User*. 2nd ed. New Jersey: Prentice Hall PTR, 1999.

[17] Lorenz, E. N.: "Deterministic Nonperiodic Flow". In: *Journal of the Atmospheric Sciences* 20.2 (1963), S. 130–141. DOI: 10.1175/1520-0469(1963)020<0130:DNF>2.0.CO;2.

[18] Lotka, A. J.: *Elements of Physical Biology*. 1925.

[19] Maciejowski, J. M.: *Predictive Control - With Constraints*. 01. Aufl. Amsterdam: Pearson Education, 2002.

[20] Naik, M. und Cochran, D.: "Nonlinear system identification using compressed sensing". In: *2012 Conference Record of the Forty Sixth Asilomar Conference on Signals, Systems and Computers (ASILOMAR)*. 2012, S. 426–430. DOI: 10.1109/ACSSC.2012.6489039.

[21] Patterson, D. W.: *Künstliche neuronale Netze: das Lehrbuch*. 1. Aufl. München [u.a.] : Prentice Hall, 1997.

[22] Pedregosa, F. u. a.: "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), S. 2825–2830.

[23] Pontryagin, L. S.: *Mathematical Theory of Optimal Processes* -. English. Boca Raton, Fla: CRC Press, 1987.

[24] Rau, M.: "Nichtlineare modellbasierte prädiktive Regelung auf Basis lernfähiger Zustandsraummodelle". type. Lehrstuhl für Elektrische Antriebssysteme, Technische Universität München.

[25] Rojas, P.: *Theorie der neuronalen Netze - Eine systematische Einführung*. 1. Aufl. Berlin Heidelberg New York: Springer-Verlag, 1993.

[26] Röbenack, K.: *Vorlesung Prozessidentifikation*. TU Dresden, Institut für Regelungs- und Steuerungstheorie, 2018.

[27] Tibshirani, R.: “Regression Shrinkage and Selection via the Lasso”. In: *Journal of the Royal Statistical Society. Series B (Methodological)* 58.1 (1996), S. 267–288. URL: http://www.jstor.org/stable/2346178.

[28] Wang, W.-X. u. a.: “Predicting Catastrophes in Nonlinear Dynamical Systems by Compressive Sensing”. In: *Phys. Rev. Lett.* 106 (15 2011), S. 154101. DOI: 10.1103/PhysRevLett.106.154101. URL: https://link.aps.org/doi/10.1103/PhysRevLett.106.154101.

[29] Wang, Y.: “A new concept using LSTM Neural Networks for dynamic system identification”. In: Mai 2017, S. 5324–5329. DOI: 10.23919/ACC.2017.7963782.

[30] Wilke, K.-T.: *Kristallzüchtung*. 2. Aufl. Frankfurt am Main: Deutsch, 1988.

[31] Winkler, J.-C.: *Forschungsprojekte*. Online; Stand 4. Dezember 2019. 2018. URL: https://tu-dresden.de/ing/elektrotechnik/rst/forschung/forschungsprojekte.