

# **Technische Universität Dresden**

Fakultät Elektrotechnik und Informationstechnik

Institut für Regelungs- und Steuerungstheorie

## **Studienarbeit**

### **Über den Einfluss hochfrequenter mechanischer Oszillationen auf das Schaltverhalten supraleitender PID-Regler auf Quantenbasis**

**Eine Fallstudie unter besonderer Berücksichtigung  
stochastischer Einflüsse**

vorgelegt von: Julius Fiedler  
geboren am: 13. Oktober 1996 in Dresden

Betreuer:	Betreuer 1
	Betreuer 2
Verantwortlicher Hochschullehrer:	Prof. Dr.-Ing. habil. Dipl.-Math. K. Röbenack
Tag der Einreichung:	2. Februar 2222

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

# Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die von mir am heutigen Tage an der Fakultät Elektrotechnik und Informationstechnik eingereichte Studienarbeit zum Thema

## **Über den Einfluss hochfrequenter mechanischer Oszillationen auf das Schaltverhalten supraleitender PID-Regler auf Quantenbasis**

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.

Pirna, 1. Januar 2016

Julius Fiedler

## **Kurzfassung**

An dieser Stelle fügen Sie bitte eine deutsche Kurzfassung ein.

## **Abstract**

Please insert the English abstract here.

# Inhaltsverzeichnis

Verzeichnis der Formelzeichen	VI
Abbildungsverzeichnis	VII
Tabellenverzeichnis	1
<b>1 Notizen</b>	<b>2</b>
1.1 UDE . . . . .	2
1.1.1 8.7.20 . . . . .	2
1.1.2 9.7.20 . . . . .	3
1.2 Sindy . . . . .	8
1.2.1 Funktionsweise der Methode . . . . .	8
1.2.2 Vergleich von SINDY-Implementierungen in Python (PySindy) und Julia (DiffEqLibrary) . . . . .	10
1.3 errors . . . . .	13

# Verzeichnis der Formelzeichen

# Abbildungsverzeichnis

2	ude fric viskos d1 03 . . . . .	6
3	ude fric viskos und haft d1 03 d2 05 . . . . .	7
4	nur viskose Reibung . . . . .	8

# Tabellenverzeichnis

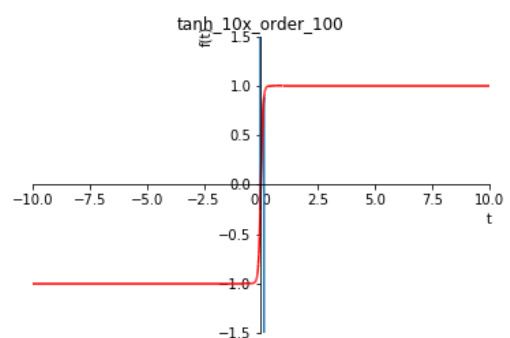
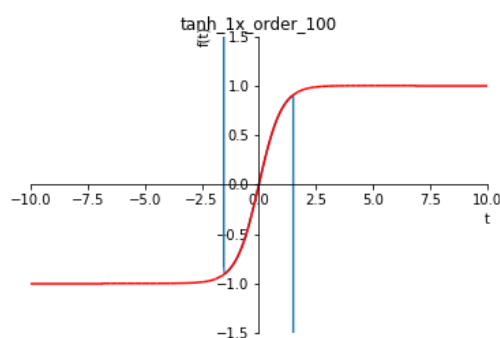
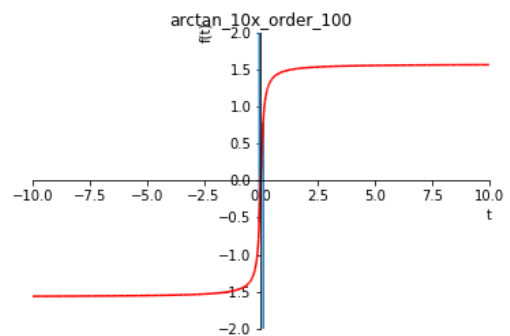
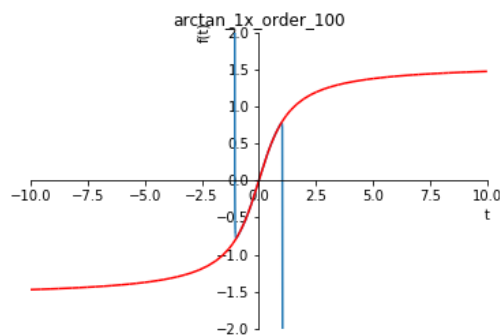
1	RMS des relativen Fehlers der Parameterschätzung, Lotka-Volterra-System	11
---	---	----



# Kapitel 1

## Notizen

### 1.1 UDE



#### 1.1.1 8.7.20

überlegung NN in julia und sindy in python machen, export erledigt  
trotzdem keine identifikation in python möglich, grund: zu wenig daten (datenfeld mit 31 daten Auflösung viel zu gering)

bei erhöhung der auflösung wird trainingszeit enorm groß NN approximiert den verlauf der ableitungen  
sindy mal mit anderen anfangswerten trainieren

### 1.1.2 9.7.20

ude keine option für multiple trajectories / threshhold???

ude + sindy

definitorische Gleichungen bekannt

dt=0.1

sindy did not converge

$$du_1 = \cos(u_3) * p_1 \quad (1.1)$$

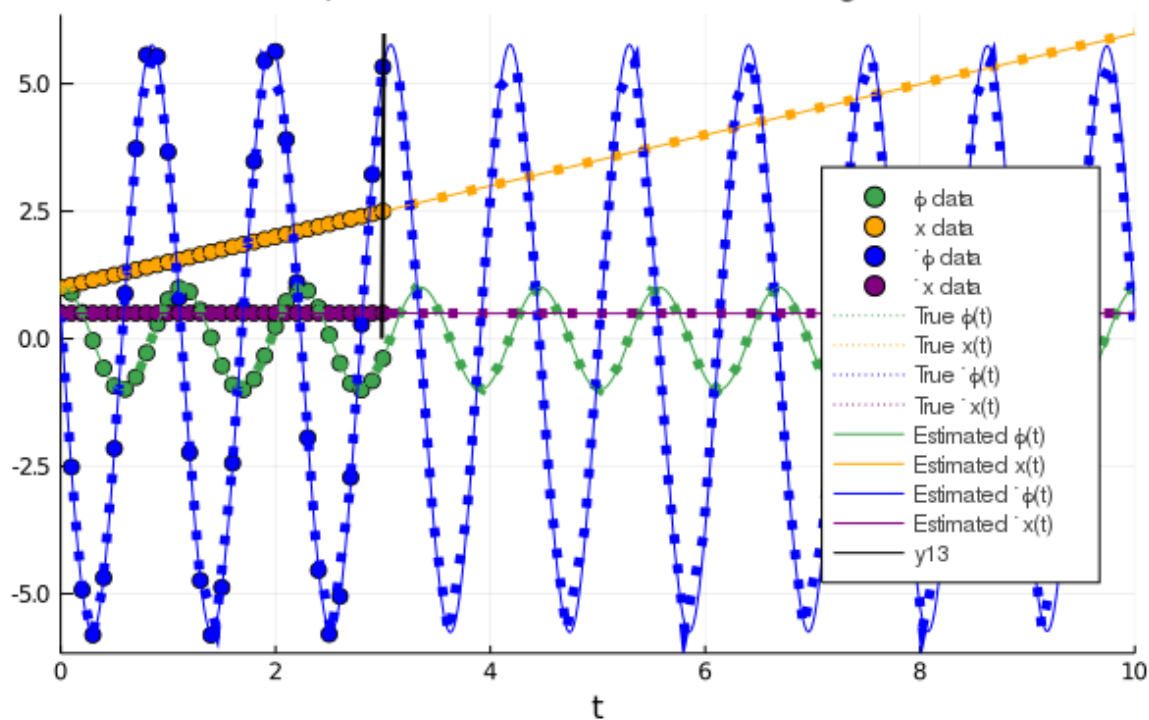
$$du_2 = \sin(u_3) * p_2 \quad (1.2)$$

$$du_3 = \sin(u_1) * p_3 + p_4 * u_1 \quad (1.3)$$

$$du_4 = \sin(u_3) * p_5 \quad (1.4)$$

parameters: Float32[0.050624948, 0.006272168, -20.755184, -13.720979, -0.01513608]

Extrapolated Fit From Short Training Data



im Vergleich: nur den Sindy Part in PySindy ausgelagert ergibt: (dieser Vergleich ist nicht sinnvoll? pysindy kennt ja die def. gl nicht)

$$\begin{aligned}
 \phi' &= 65170415.755x\dot{} + -72812822.138\sin(x\dot{}) + 2647179.524\cos(x\dot{}) \\
 x' &= 51018334.340x\dot{} + -57002163.483\sin(x\dot{}) + 2072882.755\cos(x\dot{}) \\
 \phi\dot{} &= -3.002\phi + -6.328x + 2682545588.568x\dot{} + -33.032\sin(\phi) \\
 &\quad + -1.358\sin(x) + -0.208\sin(\phi\dot{}) + -2997205703.337\sin(x\dot{}) \\
 &\quad + -0.020\cos(\phi) + -6.583\cos(x) + 109008747.059\cos(x\dot{}) \\
 x\dot{} &= 0.320x + -18300414.600x\dot{} + 20447317.167\sin(x\dot{}) + 0.344\cos(x) \\
 &\quad + -743815.199\cos(x\dot{})
 \end{aligned}$$

keine Gleichungen bekannt

dt=0.1

sindy did not converge

$$du_1 = p_1 * u_3 \quad (1.5)$$

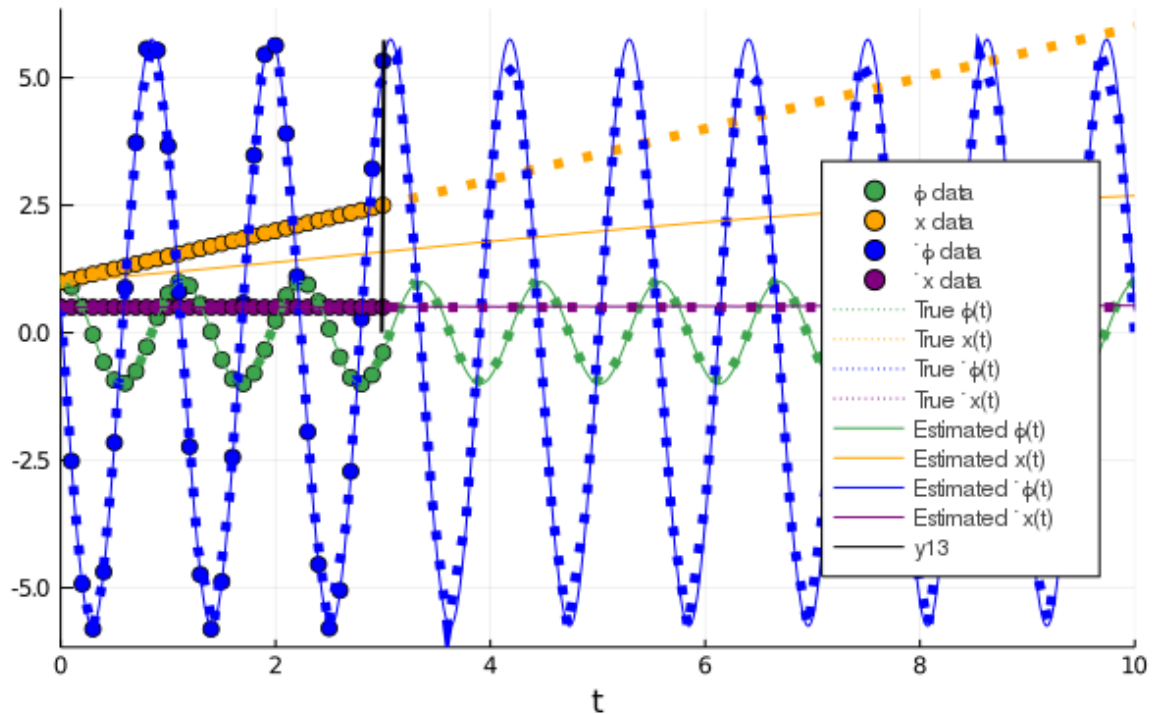
$$du_2 = \sin(u_2) * p_2 \quad (1.6)$$

$$du_3 = \sin(u_1) * p_3 + p_4 * u_1 \quad (1.7)$$

$$du_4 = \cos(u_1) * p_5 \quad (1.8)$$

parameters: Float32[0.997858, 0.20674846, -18.494001, -15.788721, 0.040644586]

### Extrapolated Fit From Short Training Data



im Vergleich: nur den Sindy Part in PySindy ausgelagert ergibt:

$$\begin{aligned}
 \phi' &= 1.111\phi + 0.648x + 1.002\phi\dot{} + -503701656.178x\dot{} + -1.228\sin(\phi) \\
 &\quad + 562779966.638\sin(x\dot{}) + -0.050\cos(\phi) + 0.676\cos(x) + -20465608.922\cos(x\dot{}) \\
 x' &= 0.833\phi + -0.031x + 450669011.982x\dot{} + -0.967\sin(\phi) \\
 &\quad + -503527393.951\sin(x\dot{}) + 0.157\cos(\phi) + -0.051\cos(x) + 18310968.307\cos(x\dot{}) \\
 \phi\dot{} &= -2.551\phi + -4.834x + -3252316739.183x\dot{} + -33.509\sin(\phi) \\
 &\quad + -0.458\sin(x) + 3633781720.861\sin(x\dot{}) + -5.066\cos(x) + -132146405.440\cos(x\dot{}) \\
 x\dot{} &= 0.674\phi + -184783222.592x\dot{} + -0.759\sin(\phi) + 206455836.816\sin(x\dot{}) + \\
 &\quad - 7507657.694\cos(x\dot{})
 \end{aligned}$$

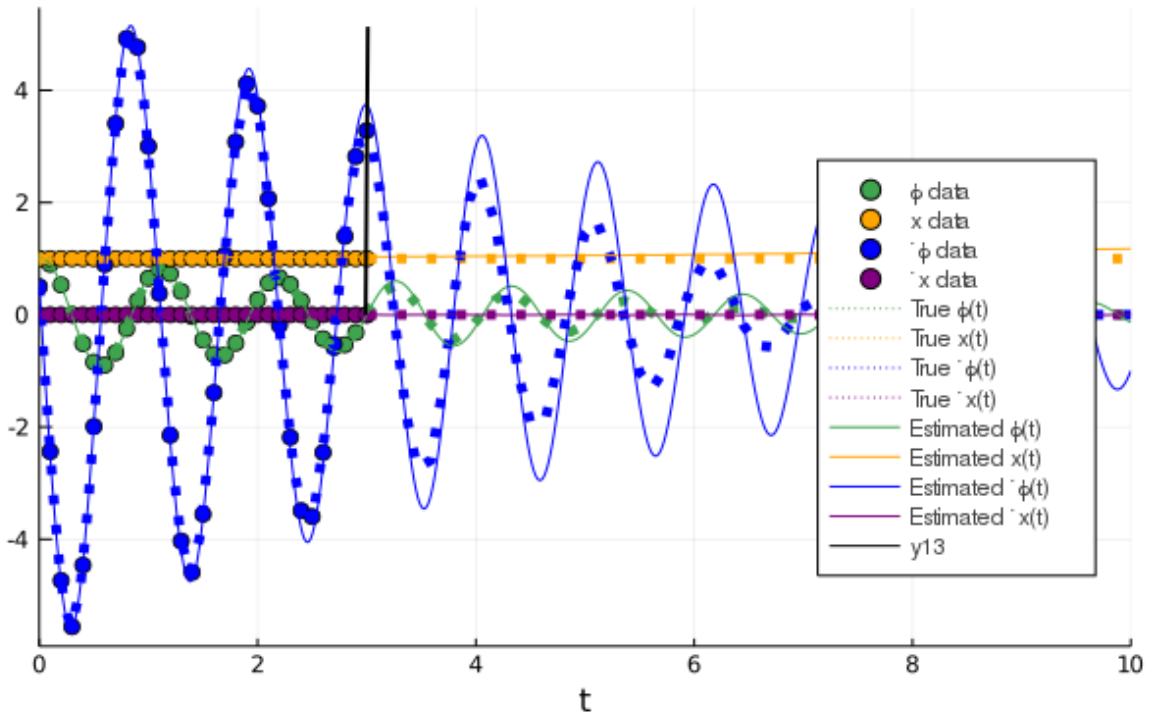
## ude reibung

tanh wie erwartet problematisch, vmtl sinnvoll tanh vorzugeben  
viskose+ haftreibung:

$$\begin{aligned}
 du_1 &= \sin(u_3) * p_1 \\
 du_2 &= \cos(u_1) * p_2 + \cos(u_3) * p_3 \\
 du_3 &= p_4 * u_3 \\
 du_4 &= \sin(u_3) * p_5
 \end{aligned}$$

parameters: Float32[-0.032978103, 0.020778582, 0.02180107, -0.30371103, -0.026243187]

## Extrapolated Fit From Short Training Data



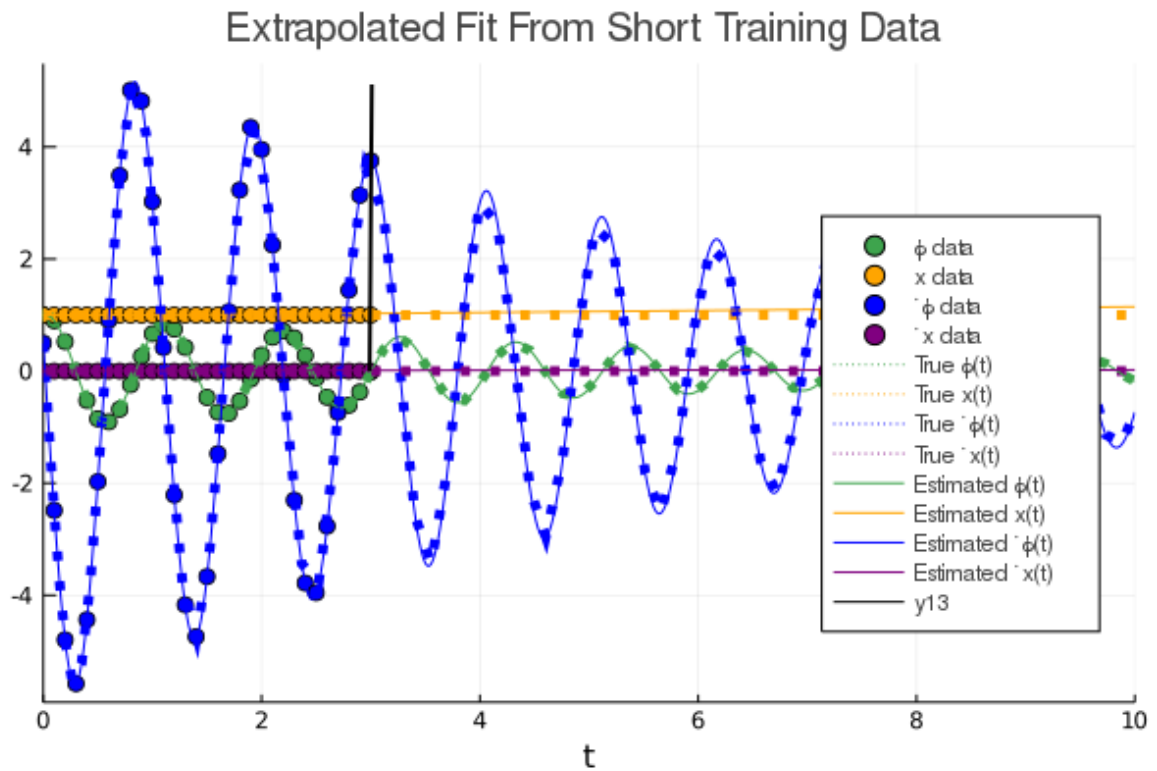


Abbildung 2 – ude fric viskos d1 03

nur viskose reibung, order 2:

$$d_1 = 0.3$$

$$d_{1,ident} = 0.2981498$$

viskose + haftreibung: order 1, tanh vorgegeben:

$$du_1 = \cos(u_1) * p_1$$

$$du_2 = \sin(u_3) * p_2$$

$$du_3 = \tanh(10u_3) * p_3 + p_4 * u_3$$

$$du_4 = \tanh(10u_3) * p_5$$

parameters: Float32[0.018757716, -0.011911368, -0.26272112, -0.35348737, -0.005909097]

## Pysindy reibung

Ansatz: Sindy für Differenz von Realem Modell mit Reibung und theoretischem Modell verwenden, siehe Bsp:

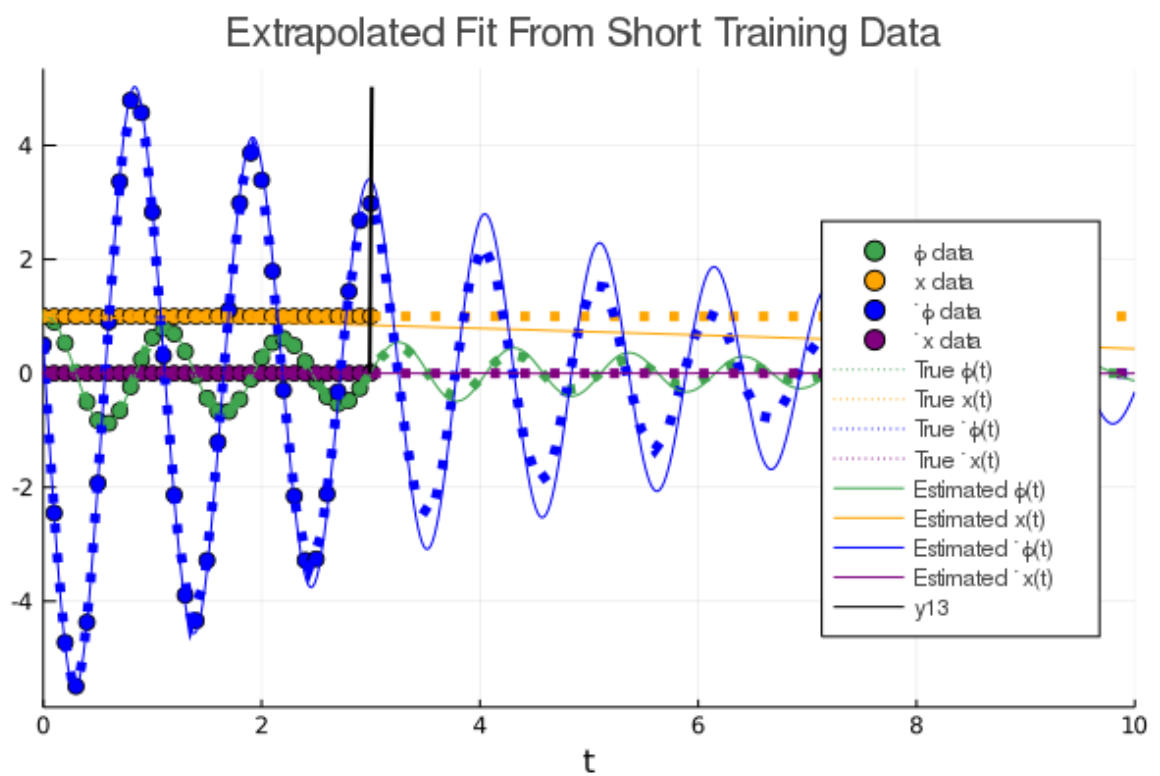


Abbildung 3 – ude fric viskos und haft d1 03 d2 05

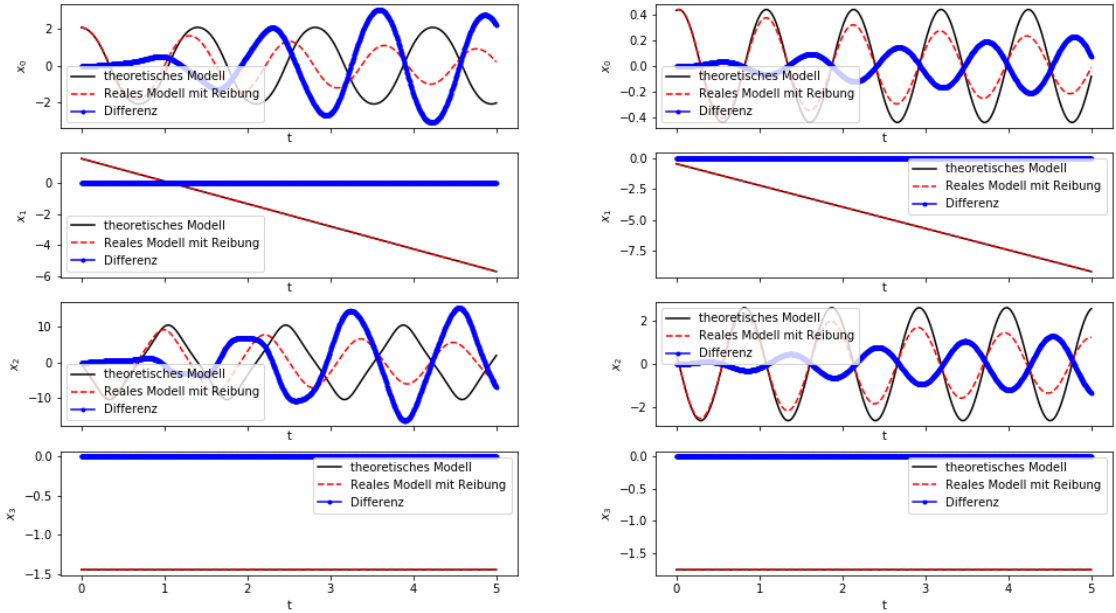


Abbildung 4 – nur viskose Reibung

keine Identifikation feststellbar

vermutung: aus Differenz geht keine exp funktion hervor, es fehlt die info über das vorherige System  
 idee: prior system knowledge integrieren indem man das bekannte teilsystem aus library funktion zur verfügung stellt

## 1.2 Sindy

### 1.2.1 Funktionsweise der Methode

Sparse Identification of Nonlinear Dynamics (SINDy) ist eine Methode, um aus Messdaten eines Systems dessen Systemdifferentialgleichungen zu schätzen.

Sei  $x(t) \in \mathbb{R}^n$  der Zustandsvektor mit  $x(t) = [x_1(t), x_2(t), \dots, x_n(t)]$ . Gesucht ist die Funktion der zeitlichen Ableitung des Zustandes  $\frac{dx(t)}{dt} = f(x(t))$ , welche auch nichtlinear sein kann. Die zugrundeliegende Überlegung der Methode ist, dass die Funktion  $f$  in einem geeigneten Raum an Basisfunktionen oft *dünn besetzt* ist. Betrachtet man beispielsweise die Funktion

$$\frac{dx}{dt} = f(x) = \begin{bmatrix} f_1(x) \\ f_2(x) \end{bmatrix} = \begin{bmatrix} 1 + 2x_1 + x_1x_2^2 \\ x_1^3 - 3x_2 \end{bmatrix} \quad (1.9)$$

so ist leicht zu erkennen, dass  $f$  in Bezug auf die Basis von Polynomen aus zwei Variablen (z.B.  $f_1(x) = \sum_{i=0}^{\infty} \sum_{j=0}^{\infty} a_{ij} x_1^i x_2^j$ ) dünn besetzt ist. Nur eine sehr geringe Zahl der Koeffizienten  $a_{ij}$  ist ungleich null. SINDy versucht nun mittels Regression diejenigen Monome auszuwählen, die die Funktion  $f$  am besten repräsentieren können.

Für die Anwendung von SINDy benötigt man die Messdaten aller Zustandsgrößen zu den Zeitpunkten  $t_1, t_2, \dots, t_m$ . Zusätzlich müssen die zeitlichen Ableitungen der Zustände an den gegebenen Zeitpunkten gegeben sein, entweder durch direkte Messung oder durch numerische Approximation. Die Daten werden wie folgt angeordnet:

$$X = \begin{bmatrix} x_1(t_1) & x_2(t_1) & \dots & x_n(t_1) \\ x_1(t_2) & x_2(t_2) & \dots & x_n(t_2) \\ \vdots & \vdots & & \vdots \\ x_1(t_m) & x_2(t_m) & \dots & x_n(t_m) \end{bmatrix} \in \mathbb{R}^{m \times n}, \quad (1.10)$$

$$\dot{X} = \begin{bmatrix} \dot{x}_1(t_1) & \dot{x}_2(t_1) & \dots & \dot{x}_n(t_1) \\ \dot{x}_1(t_2) & \dot{x}_2(t_2) & \dots & \dot{x}_n(t_2) \\ \vdots & \vdots & & \vdots \\ \dot{x}_1(t_m) & \dot{x}_2(t_m) & \dots & \dot{x}_n(t_m) \end{bmatrix} \in \mathbb{R}^{m \times n}. \quad (1.11)$$

Nun muss man die Bibliothek  $\Theta$  an Funktionen konstruieren, durch welche  $f$  dargestellt werden soll. Hier ist es günstig, wenn man bereits weiß, welche Funktionsklassen in  $f$  vorkommen, um die Bibliothek geeignet auszulegen. Die Spalten der Bibliotheksmatrix repräsentieren die gewählten Ansatzfunktionen, angewendet auf die Datenmatrix  $X$

$$\Theta(X) = \begin{bmatrix} \left| \begin{array}{c} \theta_1(X) \\ \theta_2(X) \\ \dots \\ \theta_\ell(X) \end{array} \right| \end{bmatrix} \in \mathbb{R}^{m \times L}. \quad (1.12)$$

Wählt man beispielsweise für  $\theta_1$  die Sinusfunktion und für  $\theta_2$  Monome zweiten Grades so ergeben sich

$$\theta_1(X) = \begin{bmatrix} \left| \begin{array}{c} \sin(x_1(t)) \\ \sin(x_2(t)) \\ \dots \\ \sin(x_n(t)) \end{array} \right| \end{bmatrix}, \quad (1.13)$$

$$\theta_2(X) = \begin{bmatrix} \left| \begin{array}{c} x_1(t)^2 \\ x_1(t)x_2(t) \\ \dots \\ x_2(t)^2 \\ x_2(t)x_3(t) \\ \dots \\ x_n^2(t) \end{array} \right| \end{bmatrix}, \quad (1.14)$$

wobei die Rechenoperationen alle elementweise zu lesen sind.

Gesucht sind nun Linearkombinationen von Bibliotheksfunktionen, sodass gilt

$$f_i(x) = \Theta(x^T) \xi_i. \quad (1.15)$$

Fasst man alle  $\xi_i$  in eine Koeffizientenmatrix  $\Xi \in \mathbb{R}^{L \times n}$  zusammen, so ergibt sich das zu lösende Problem zu

$$\dot{X} \approx \Theta(X) \Xi. \quad (1.16)$$



In der Praxis sollte dieses Gleichungssystem überbestimmt sein. Das bedeutet, dass die Anzahl der Messzeitpunkte  $m$  (die Anzahl der Zeilen) größer ist als die Anzahl der Einträge in  $\Xi \in \mathbb{R}^{L \times n}$  (Anzahl der Unbekannten). Um das SINDy-Problem zu lösen, wird die Methode der kleinsten Quadrate angewendet. Durch aufstellen der Pseudo-Inversen ergibt sich

$$\Xi = \left( \Theta(X)^T \Theta(X) \right)^{-1} \Theta(X)^T \dot{X} \quad (1.17)$$

Im Programm implementiert: mit

$$\Xi_{\text{neu}} = \left( \Theta(X)^T \Theta(X) + I \right)^{-1} \left( \Theta(X)^T \dot{X} + \Xi_{\text{alt}} \right) \quad (1.18)$$

iterieren und am Ende

initial guess: Solves the equation  $ax = b$  by computing a vector  $x$  that minimizes the squared Euclidean 2-norm  $\|b - ax\|_2^2$ . The equation may be under-, well-, or over-determined (i.e., the number of linearly independent rows of  $a$  can be less than, equal to, or greater than its number of linearly independent columns). If  $a$  is square and of full rank, then  $x$  (but for round-off error) is the exact solution of the equation.

## 1.2.2 Vergleich von SINDY-Implementierungen in Python (PySindy) und Julia (DiffEqLibrary)

System: Volterra DGL

$$\dot{x} = \alpha x + \beta xy \quad (1.19)$$

$$\dot{y} = \gamma y + \delta xy \quad (1.20)$$

mit  $[\alpha, \beta, \gamma, \delta] = [1.3, -0.9, -1.8, 0.8]$

Zeitspanne 3s

Schrittweite 0.1s

Polynomial Library Order 2

nicht normalisiert

maximale Iterationen: 30

Threshold 0.2

Datenerzeugung durch Lösen der DGL mittels DGL-Solver, Verwendung der exakt gleichen Datensätze in beiden Algorithmen

Fall 1: exakte Ableitungen bekannt

Parameter	Nominal	DiffEq	PySindy
$\alpha$	1.3	1.291	1.300
$\beta$	-0.9	-0.850	-0.900
$\gamma$	-1.8	-1.801	-1.800
$\delta$	0.8	0.753	0.800

Fall 2: Zentraldifferenz für Ableitungen verwendet

Parameter	Nominal	DiffEq	PySindy
$\alpha$	1.3	1.301	1.302
$\beta$	-0.9	-0.683	-0.903
$\gamma$	-1.8	-1.816	-1.802
$\delta$	0.8	0.614	0.798

Fall 3: Ableitungen durch Neuronales Netz gelernt, gleiche Auflösung

Parameter	Nominal	DiffEq	PySindy
$\alpha$	1.3	1.438	1.289
$\beta$	-0.9	-0.226	-0.868
$\gamma$	-1.8	-1.814	-1.780
$\delta$	0.8	0.386	0.721

Fall 4: Ableitungen durch Neuronales Netz gelernt, Schrittweite 0.01s (NN auf wenig Daten trainiert, viele Daten abgerufen)

Parameter	Nominal	DiffEq	PySindy
$\alpha$	1.3	1.471	1.294
$\beta$	-0.9	-0.353	-0.887
$\gamma$	-1.8	-1.818	-1.786
$\delta$	0.8	0.473	0.744

Fall 5: Ableitungen durch Neuronales Netz gelernt, Schrittweite 0.001s (NN auf wenig Daten trainiert, viele Daten abgerufen)

Parameter	Nominal	DiffEq	PySindy
$\alpha$	1.3	1.477	1.294
$\beta$	-0.9	-0.367	-0.889
$\gamma$	-1.8	-1.820	-1.786
$\delta$	0.8	0.480	0.748

	DiffEq	PySindy	Sindy naiv
Exakte Ableitung bekannt	0.040883	$5.013162 \cdot 10^{-7}$	$2.608947 \cdot 10^{-6}$
Ableitung über Zentraldifferenz	0.167287	0.002317	0.002318
Ableitung durch NN, sample interval 0.1s	0.212189	0.025098	0.025101
Ableitung durch NN, sample interval 0.01s	0.135958	0.012395	0.012304
Ableitung durch NN, sample interval 0.001s	0.126144	0.010833	0.010838

**Tabelle 1** – RMS des relativen Fehlers der Parameterschätzung, Lotka-Volterra-System

volterra	nominal	zentral	NN	NN x10	NN x100
PySINDy abs. Fehler	0.0	0.00141	0.0121	0.00591	0.00516
PySINDy rel. Fehler	0.0	0.00232	0.0251	0.0124	0.01083
DiffEq abs. Fehler	0.02013	0.08252	0.10629	0.06855	0.06377
DiffEq rel. Fehler	0.04088	0.16729	0.21219	0.13596	0.12614
SINDy naiv abs. Fehler	0.0	0.00141	0.0121	0.00587	0.00516
SINDy naiv rel. Fehler	0.0	0.00232	0.0251	0.0123	0.01084
lorenz	nominal	zentral			
PySINDy abs. Fehler	0.0	0.03154			
PySINDy rel. Fehler	0.0	0.0109*			
DiffEq abs. Fehler	3.0e-5	0.27847			
DiffEq rel. Fehler	3.0e-5	0.04201*			
SINDy naiv abs. Fehler	0.0	0.0312			
SINDy naiv rel. Fehler	0.0	0.0109			
roessler	nominal	zentral	NN	NN x10	
PySINDy abs. Fehler	0.0	0.00059	7.08376	0.37128	
PySINDy rel. Fehler	0.0	0.00082*	34.4256*	3.71507*	
DiffEq abs. Fehler	0.0004	0.05572	7.08713	0.40965	
DiffEq rel. Fehler	0.0022	0.20196*	34.41761*	3.9332*	
SINDy naiv abs. Fehler	0.0	0.00059	7.08376	0.37118	
SINDy naiv rel. Fehler	0.0	0.00082	34.4256*	3.71507*	
volterra	nominal	zentral	NN	NN x10	NN x100
PySINDy abs. Fehler	0.0	0.00141	0.0121	0.00591	0.00516
DiffEq abs. Fehler	0.02013	0.08252	0.10629	0.06855	0.06377
SINDy naiv abs. Fehler	0.0	0.00141	0.0121	0.00587	0.00516
PySINDy rel. Fehler	0.0	0.00232	0.0251	0.0124	0.01083
DiffEq rel. Fehler	0.04088	0.16729	0.21219	0.13596	0.12614
SINDy naiv rel. Fehler	0.0	0.00232	0.0251	0.0123	0.01084
lorenz	nominal	zentral			
PySINDy abs. Fehler	0.0	0.03154			
DiffEq abs. Fehler	3.0e-5	0.27847			
SINDy naiv abs. Fehler	0.0	0.0312			
PySINDy rel. Fehler	0.0	0.0109*			
DiffEq rel. Fehler	3.0e-5	0.04201*			
SINDy naiv rel. Fehler	0.0	0.0109			
roessler	nominal	zentral	NN	NN x10	
PySINDy abs. Fehler	0.0	0.00059	7.08376	0.37128	
DiffEq abs. Fehler	0.0004	0.05572	7.08713	0.40965	
SINDy naiv abs. Fehler	0.0	0.00059	7.08376	0.37118	
PySINDy rel. Fehler	0.0	0.00082*	34.4256*	3.71507*	
DiffEq rel. Fehler	0.0022	0.20196*	34.41761*	3.9332*	
SINDy naiv rel. Fehler	0.0	0.00082	34.4256*	3.71507*	

anzahl der iterationen in pysindy relevant, wenn ident mit vorwissen, bei zentraldiff und lin term bekannt und  $maxiter = 1$  kommt lin term hinzu, bei  $maxiter = 2$  nicht  
bei test mit NN fällt auf, dass sindy ergebnisse stark schwanken weil NN teilweise schlecht

problem bei relativem Fehler: wenn  $p_{nom} = 0$  und  $p_{ident!} = 0$  wie darstellen?

$\Xi_{alt}$  aus PySindy zu streichen macht den algorithmus kaputt

sindy naiv ist ein ganz klein bissl schlechter als PySindy

naiv ein wenig schlechter als pysindy?

abtastrate für zb lorenz von großer bedeutung für identifizierbarkeit, wie vergleichbar machen?

fehler in zentraldiff/NN festhalten, max oder durchschnitt?

sindy naiv: nach 2. mkq mit ausgewählten spalten nochmal alle coeff nullsetzen? in pysindy wird das nicht gemacht siehe roessler zentral  $pi_{zentral}[2][8] = -6 * 10^{-5}$  wird aber nicht angezeigt

sparse regression ansatz schwierig bei systemen mit kleinen parametern tb roessler  
vgl bei "high res" von X aus NN vs X aus odesolver

## 1.3 errors

Ude:

*AssertionError : length(b) == length(variables(b)) in unknown\_sys :*

wenn  $\lambda$  so gewählt, dass ganze Zeilen rausfallen

nützliche Befehle: