

# **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	VIII
1 Notizen	IX
1.1 UDE	IX
1.1.1 8.7.20	IX
1.1.2 9.7.20	X
1.2 Sindy	XV
1.2.1 Funktionsweise der Methode	XV
1.2.2 Vergleich Sindy in Python und Julia	XVII
1.3 errors	XVIII

# Verzeichnis der Formelzeichen

# Abbildungsverzeichnis

2	ude fric viskos d1 03 . . . . .	XIII
3	ude fric viskos und haft d1 03 d2 05 . . . . .	XIV
4	nur viskose Reibung . . . . .	XV

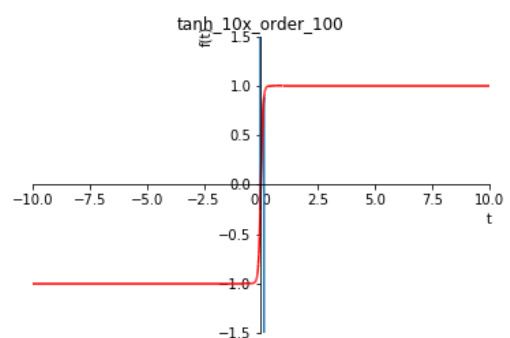
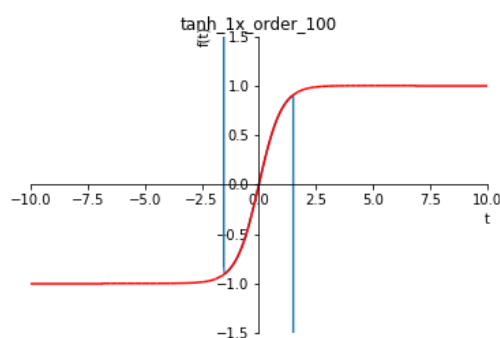
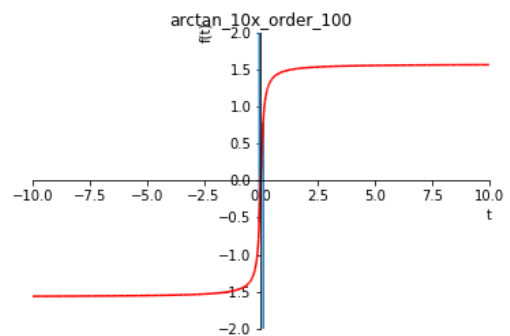
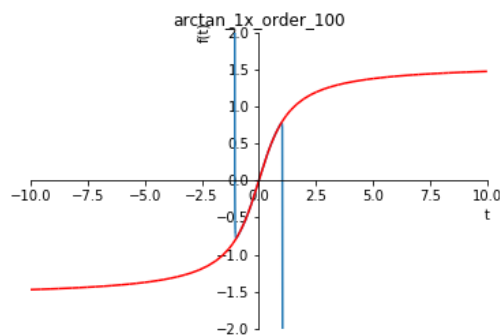
# Tabellenverzeichnis



# 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

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

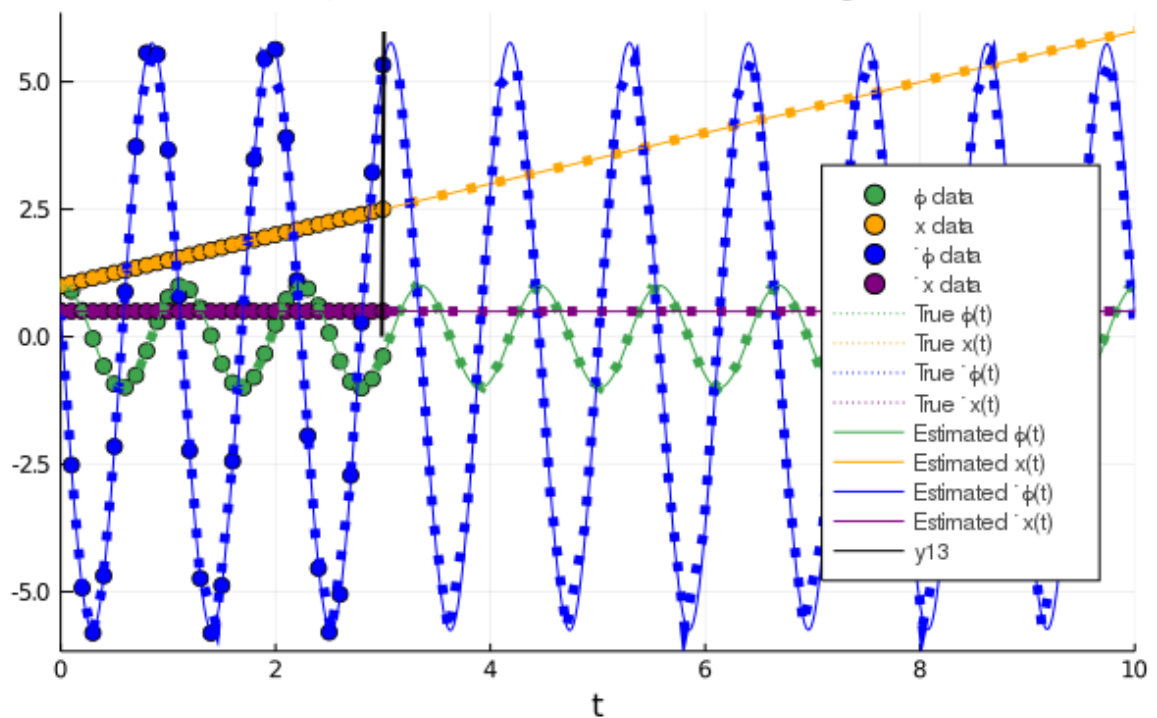
$$du_2 = \sin(u_3) * p_2 \quad (1.2)$$

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

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

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

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

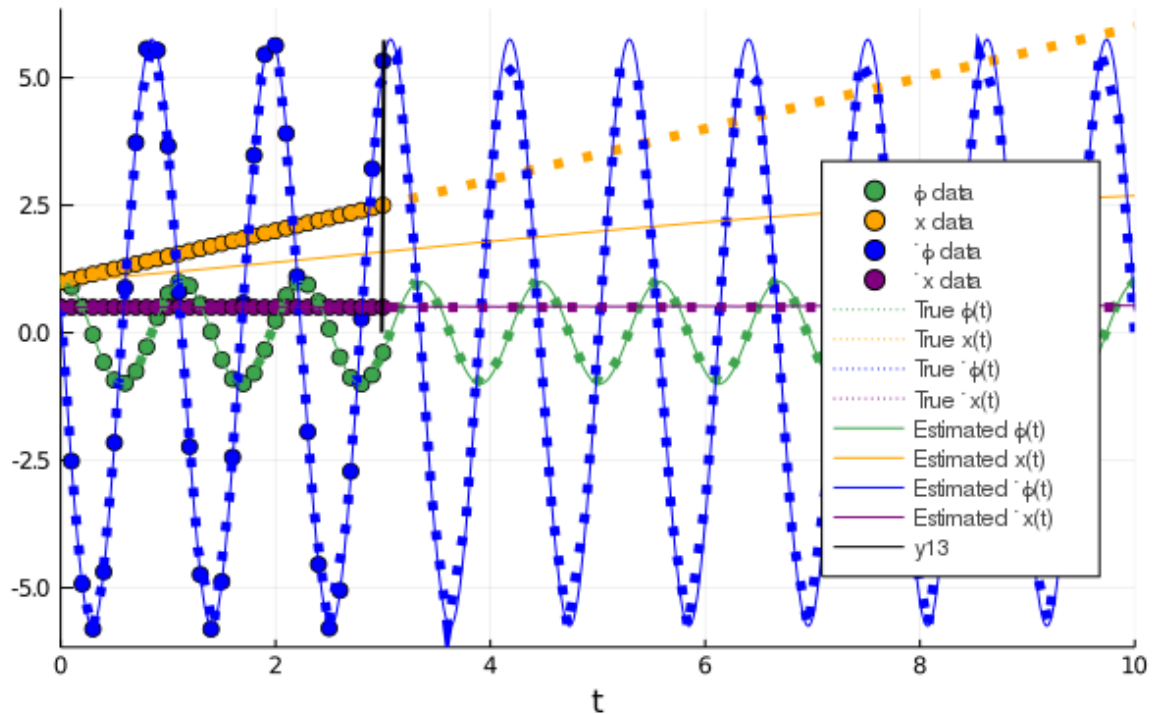
$$du_2 = \sin(u_2) * p_2 \quad (1.6)$$

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

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

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:

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

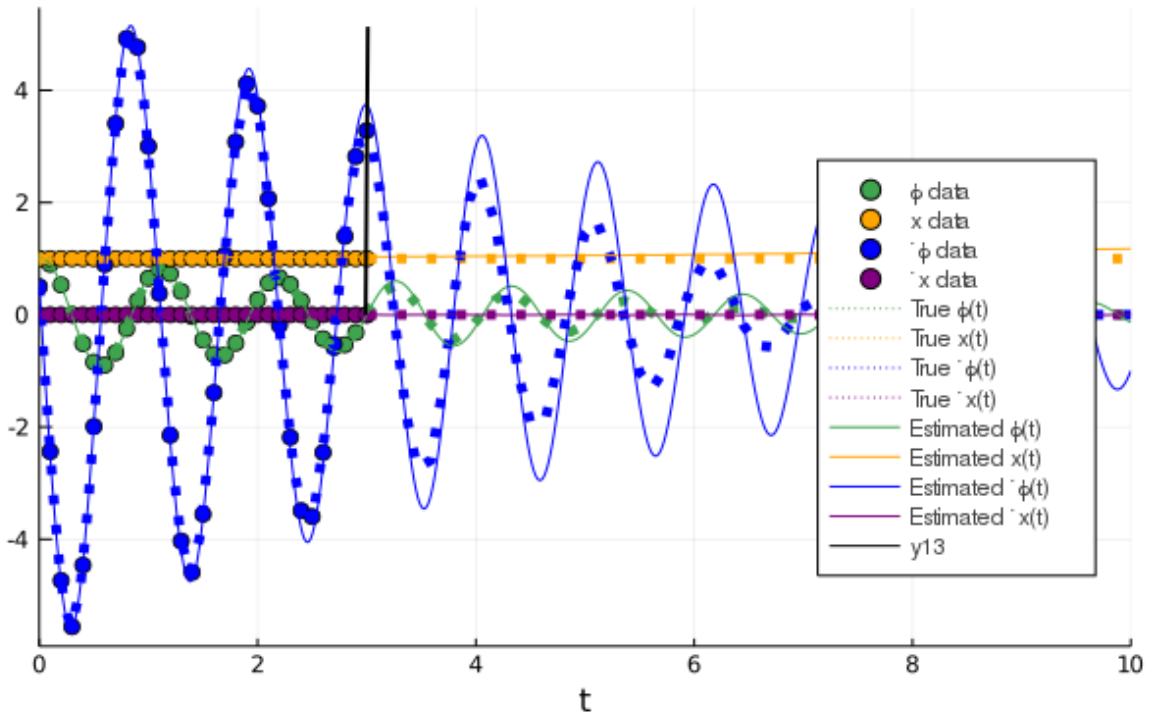
## ude reibung

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



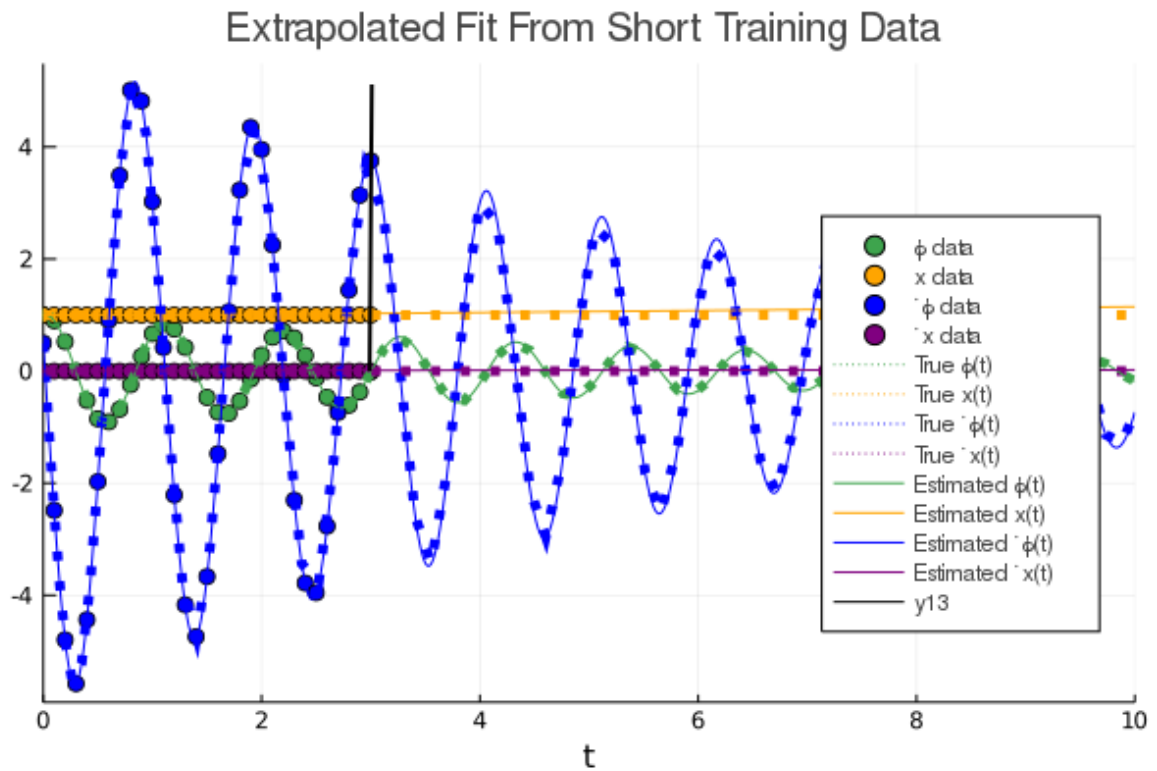


Abbildung 2 – ude fric viskos d1 03

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:

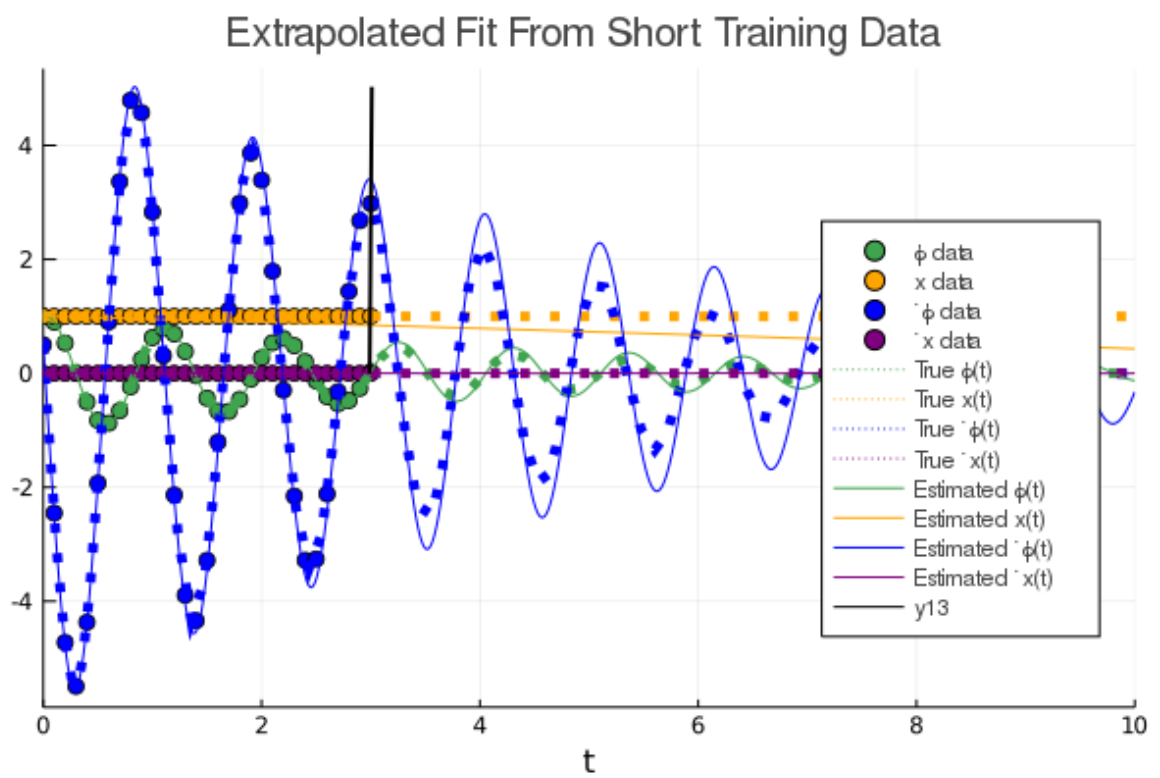


Abbildung 3 – ude fric viskos und haft d1 03 d2 05

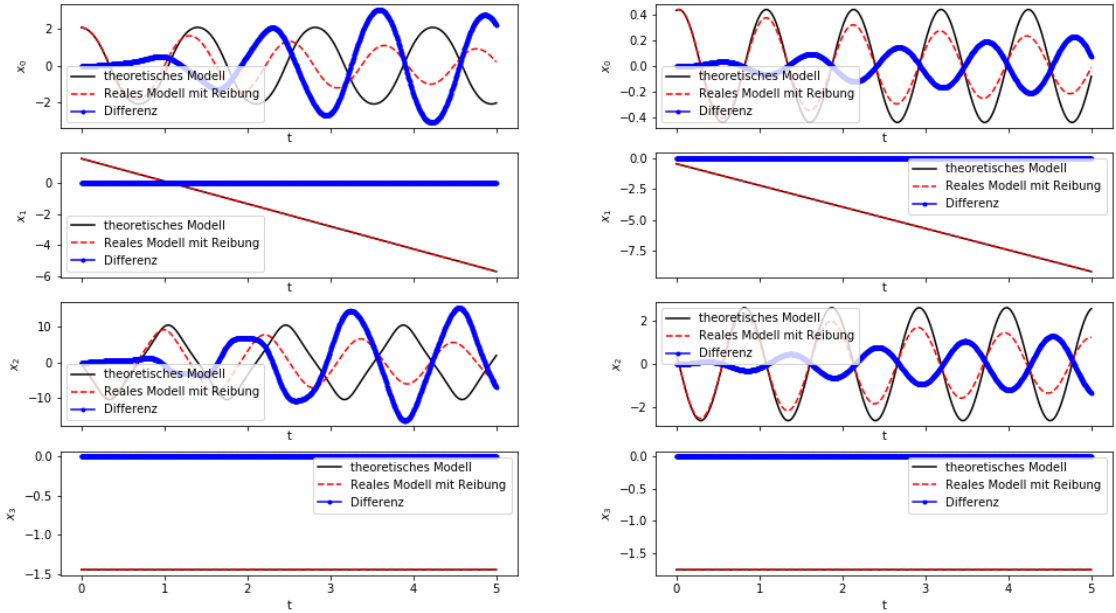


Abbildung 4 – nur viskose Reibung

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 die zu lösende Gleichung zu

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



Benötigt werden die Zeitverläufe aller Zustände  $x(t) \in \mathbb{R}^n$  Erzeugen der Library  
 Zeitverläufe X in Library functions auswerten (geg: x[], y[], library berechnet ident, x, y,  
 x\*y, x\*\*2 usw an jedem zeitpunkt des arrays)

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 Sindy in Python und Julia

System: Volterra (julia : normalize true, maxiter 10000)

Zeitspanne: 3s

Schrittweite: 0.1s

Julia:

$$\begin{aligned}\dot{x} &= -0.057677828y^2 + 0.9879066x \\ \dot{y} &= -1.5857016y\end{aligned}$$

Python:

$$\begin{aligned}\dot{x} &= 0.973x \\ \dot{y} &= -2.103y + 1.561xy\end{aligned}$$

Zeitspanne: 3s

Schrittweite: 0.01s

Julia:

$$\begin{aligned}\dot{x} &= -0.0028148904y^2 + 1.2928892x - 0.8812496xy \\ \dot{y} &= -1.5789621y\end{aligned}$$

Python:

$$\begin{aligned}\dot{x} &= 1.293x + -0.909xy \\ \dot{y} &= -1.821y + 0.837xy\end{aligned}$$

Zeitspanne: 3s

Schrittweite: 0.001s

Julia:

$$\begin{aligned}\dot{x} &= 1.29908x - 0.9018026xy \\ \dot{y} &= -0.3144052 - 0.1010906y^2 - 0.4039814y - 1.0950719xy\end{aligned}$$

Python:

$$\begin{aligned}\dot{x} &= 1.299x + -0.901xy \\ \dot{y} &= -1.802y + 0.803xy\end{aligned}$$

## 1.3 errors

Ude:

*AssertionError : length(b) == length(variables(b))inunknownsys :*

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

nützliche Befehle: