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

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Der SINDy-Algorithmus

1.1 Idee

Sparse Identification of Nonlinear Dynamics (SINDy) ist eine Methode, um aus den Messdaten eines Systems auf dessen Systemdifferentialgleichungen zu schließen. Die zugrundeliegende Annahme ist, dass die zu identifizierende Funktion in einem geeigneten Raum an Basisfunktionen *dünn besetzt* ist. Betrachtet man beispielsweise die Funktion

$$\frac{\mathrm{d}x}{\mathrm{d}t} = f(x) = \begin{bmatrix} f_1(x) \\ f_2(x) \end{bmatrix} = \begin{bmatrix} 1 + 2x_1 + x_1 x_2^2 \\ x_1^3 - 3x_2 \end{bmatrix}, \tag{1.1}$$

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. Die dem Algorithmus zur Verfügung gestellten Basisfunktionen werden zu einer Bibliothek zusammengefasst. Hieraus werden mittels Regression diejenigen Ansatzfunktionen ausgewählt, deren Linearkombination die Funktion f am besten repräsentiert. Es ist einsichtig, dass die Auswahl der Bibliotheksfunktionen eine entscheidende Rolle für den Erfolg der Methode spielt. Daher ist es günstig, wenn man bereits weiß, welche Funktionsklassen zu identifizieren sind, um die Bibliothek geeignet auszulegen. Wie der Methodenname suggeriert, können mit SINDy auch nichtlineare Funktionen identifiziert werden.

1.2 Funktionsweise der Methode

Sei $f\left(x(t)\right) = \frac{\mathrm{d}x(t)}{\mathrm{d}t}$ das zu identifizierende Differentialgleichungssystem mit dem Zustandsvektor $x(t) = [x_1(t), x_2(t), ..., x_n(t)]^T \in \mathbb{R}^n$. Für die Anwendung von SINDy benötigt man die Messdaten aller Zustandsgrößen zu den Zeitpunkten $t_1, t_2, ..., t_m$. 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:

$$X = \begin{bmatrix} x_1(t_1) & x_2(t_1) & \dots & x_n(t_1) \\ x_1(t_2) & x_2(t_2) & \dots & x_n(t_2) \\ \vdots & \vdots & & \vdots \\ x_1(t_m) & x_2(t_m) & \dots & x_n(t_m) \end{bmatrix} \in \mathbb{R}^{m \times n}, \tag{1.2}$$

$$\dot{X} = \begin{bmatrix} \dot{x}_1(t_1) & \dot{x}_2(t_1) & \dots & \dot{x}_n(t_1) \\ \dot{x}_1(t_2) & \dot{x}_2(t_2) & \dots & \dot{x}_n(t_2) \\ \vdots & \vdots & & \vdots \\ \dot{x}_1(t_m) & \dot{x}_2(t_m) & \dots & \dot{x}_n(t_m) \end{bmatrix} \in \mathbb{R}^{m \times n}. \tag{1.3}$$

Nun muss man die Bibliothek Θ an Funktionen konstruieren, durch welche f dargestellt werden soll. Die Spalten der Bibliotheksmatrix repräsentieren die gewählten Ansatzfunktionen, angewendet auf die Datenmatrix X

$$\Theta(X) = \begin{bmatrix} | & | & & | \\ \theta_1(X) & \theta_2(X) & \dots & \theta_\ell(X) \\ | & | & & | \end{bmatrix} \in \mathbb{R}^{m \times L}. \tag{1.4}$$

Wählt man beispielsweise für θ_1 die Sinusfunktion und für θ_2 Monome zweiten Grades so ergeben sich

$$\theta_1(X) = \begin{bmatrix} | & | & & | \\ \sin(x_1(t)) & \sin(x_2(t)) & \dots & \sin(x_n(t)) \\ | & | & & | \end{bmatrix}, \tag{1.5}$$

$$\theta_2(X) = \begin{bmatrix} | & | & & | & | & & | \\ x_1(t)^2 & x_1(t)x_2(t) & \dots & x_2(t)^2 & x_2(t)x_3(t) & \dots & x_n^2(t) \\ | & | & & | & | & & | \end{bmatrix}, \tag{1.6}$$

wobei die Rechenoperationen alle elementweise zu lesen sind.
Gesucht sind nun Linearkombinationen von Bibliotheksfunktionen, sodass gilt

$$f_i(x) = \Theta(x^T)\xi_i. \tag{1.7}$$

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

$$\dot{X} \approx \Theta(X)\Xi. \tag{1.8}$$

[1]
Der SINDy-Algorithmus wurde sowohl in Python durch die Bibliothek PySindy [2] als auch in Julia durch die Bibliothek DataDrivenDiffEq implementiert. Im Verlauf dieser Arbeit ist ein vereinfachter SINDy-Algorithmus entstanden, der mit den existierenden Implementationen verglichen werden soll.

1.3 Lösen des Minimierungsproblems

Das Lösen des Gleichungssystems 1.8 ist mathematisch gesehen die Kernaufgabe des Algorithmus. Das Gleichungssystem besitzt $m \cdot n$ Gleichungen und $L \cdot n$ Unbekannte. In der Regel ist dieses Gleichungssystem überbestimmt, da die Anzahl der Ansatzfunktionen überschaubar sein sollte, während die Zahl der Messungen deutlich größer sein sollte. Damit kann die Lösung über die Moore-Penrose-Inverse von $\Theta(X)$ berechnet werden.

$$\Xi = \Theta^{+}\dot{X} \tag{1.9}$$

Diese minimiert den Fehler

$$\epsilon = \left\| \Theta(X)\Xi - \dot{X} \right\|_2. \tag{1.10}$$

Um die Forderung nach einer dünnbesetzten Matrix umzusetzen, werden anschließend alle Koeffizienten, deren Betrag unter einem festgelegten Grenzwert liegt, zu Null gesetzt.

$$\Xi_{\text{dünn, ij}} = \begin{cases} 0 & |\Xi_{\text{ij}}| < \lambda \\ \Xi_{\text{ij}} & \text{sonst} \end{cases}, \quad 1 \leq i \leq L, \quad 1 \leq j \leq n \tag{1.11}$$

Jeder nicht negative Koeffizient $\Xi_{\text{dünn, ij}}$ repräsentiert eine im Differentialgleichungssystem vorkommende Ansatzfunktion. Allerdings sind die Koeffizienten $\Xi_{\text{dünn}}$ fehlerhaft, da sie auf Basis der nicht vorkommenden Funktionen berechnet wurden. Es bietet sich an, die Koeffizientenmatrix Ξ erneut zu berechnen, unter Nutzung des Wissens über welche Ansatzfunktionen in der Differentialgleichung (DGL) vorkommen.

Für jede Zeile der DGL wählt man diejenigen Spalten der Bibliotheksmatrix aus, die für Funktionen stehen, die nach 1.11 in dieser Zeile vorkommen. Dann kann man die Koeffizientenmatrix spaltenweise wie folgt berechnen (der Index a steht für Auswahl, die Indices der θ sind willkürlich gewählt):

$$\dot{X}_i = \begin{pmatrix} | \\ \dot{x}_i(t) \\ | \end{pmatrix} \in \mathbb{R}^m \tag{1.12a}$$

$$\Theta_{\text{a}}(X) = \begin{bmatrix} | & | & | & \\ \theta_2(X) & \theta_5(X) & \theta_7(X) & \dots \\ | & | & | & \end{bmatrix} \in \mathbb{R}^{m \times l}. \tag{1.12b}$$

$$\hat{\Xi}_i = \xi_i \in \mathbb{R}^l = \Theta_{\text{a}}^{+}\dot{X}_i \tag{1.12c}$$

Die Spalten $\hat{\Xi}_i$ können zu einer Matrix $\hat{\Xi}$ zusammengesetzt werden. Anschließend muss diese auf $\Xi \in \mathbb{R}^{L \times n}$ vergrößert und an den entsprechenden Stellen mit Nullen aufgefüllt werden:

$$\Xi_{\text{neu, ij}} = \begin{cases} 0 & |\Xi_{\text{ij}}| < \lambda \\ \hat{\Xi}_{\text{kj}}, 1 \leq k \leq l & \text{sonst} \end{cases}. \tag{1.13}$$

Allerdings besteht die Möglichkeit, dass Ξ_{neu} durch die erneute Berechnung Einträge unterhalb des Grenzwertes besitzt. Daher werden die Schritte 1.11, 1.12c und 1.13 solange wiederholt, bis in 1.11 $\Xi_{\text{dünn}} = \Xi$ gilt, also bis die Koeffizientenmatrix nicht mehr verändert wird. Es werden somit iterativ immer mehr Ansatzfunktionen ausgeschlossen. Damit gilt am Ende

$$\dot{X} \approx \Theta(X)\Xi \tag{1.14a}$$

und somit

$$\dot{x} = f(x) \approx \Xi^T \left(\Theta(x^T)\right)^T, \tag{1.14b}$$

wobei Ξ dünn besetzt ist.

Kapitel 2

Vergleich der Implementationen

Für den Vergleich der Implementationen werden drei verschiedene Differentialgleichungssystem betrachtet, deren Parameter es zu identifizieren gilt.

Das Lotka-Volterra-System

$$\begin{aligned} \dot{x} &= \alpha x + \beta xy \\ \dot{y} &= \gamma y + \delta xy \end{aligned} \tag{2.1}$$

mit den Nominalparametern

$$p_{\mathrm{LV}} = \begin{pmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{pmatrix} = \begin{pmatrix} 1.3 \\ -0.9 \\ 0.8 \\ -1.8 \end{pmatrix}. \tag{2.2}$$

Das Lorenz-System

$$\begin{aligned} \dot{x} &= \alpha(y - x) \\ \dot{y} &= x(\beta - z) - y \\ \dot{z} &= xy - \gamma z \end{aligned} \tag{2.3}$$

mit den Nominalparametern

$$p_{\mathrm{L}} = \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} = \begin{pmatrix} 10 \\ 28 \\ \frac{8}{3} \end{pmatrix}. \tag{2.4}$$

Das Rössler-System

$$\begin{aligned} \dot{x} &= -y - z \\ \dot{y} &= x + \alpha y \\ \dot{z} &= \beta + (x - c)z \end{aligned} \tag{2.5}$$

mit den Nominalparametern

$$p_{\mathrm{R}} = \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} = \begin{pmatrix} 0.2 \\ 0.1 \\ 5.3 \end{pmatrix}. \tag{2.6}$$

Ausschlaggebend für die Güte des jeweiligen Algorithmus ist an erster Stelle die Genauigkeit des Ergebnisses, an zweiter Stelle die Rechenzeit. Um die Genauigkeit des Algorithmus vergleichen zu können, wird ein Fehler ϵ_r definiert. $P \in \mathbb{R}^{L\times n}$ bezeichnet die nominalen und $Q \in \mathbb{R}^{L\times n}$ die identifizierten Parameter eines Systems.

$$R \in \mathbb{R}^{L\times n}, \quad R_{ij} = \begin{cases} \frac{P_{ij}-Q_{ij}}{P_{ij}} & P_{ij} \neq 0 \\ \frac{P_{ij}-Q_{ij}}{Q_{ij}} & P_{ij} = 0 \wedge Q_{ij} \neq 0\,, \\ 0 & \text{sonst} \end{cases} \quad 1 \leq i \leq L, \quad 1 \leq j \leq n \tag{2.7}$$

$$\epsilon_r = \frac{1}{\sqrt{Ln}} \|R\|_2 \tag{2.8}$$

Für jedes System werden die zeitlichen Verläufe der Zustände durch einen Differentialgleichungslöser numerisch ermittelt. Die Ableitungsverläufe werden sowohl exakt vorgegeben, als auch über die Zentraldifferenz angenähert. Dabei wird der Einfluss folgender Parameter auf das Identifikationsergebnis untersucht:

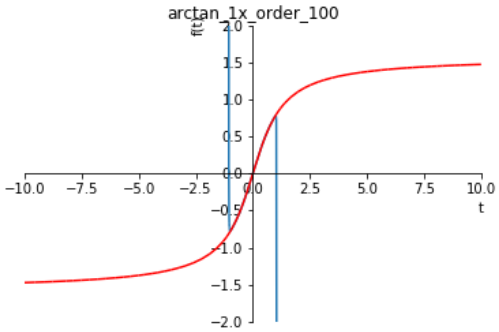
- die Simulationszeit t des DGL-Lösers
- die Schrittweite dt des DGL-Lösers
- die Verwendung mehrerer Datenreihen auf einmal
- die Gestaltung der Bibliothek aus Ansatzfunktionen

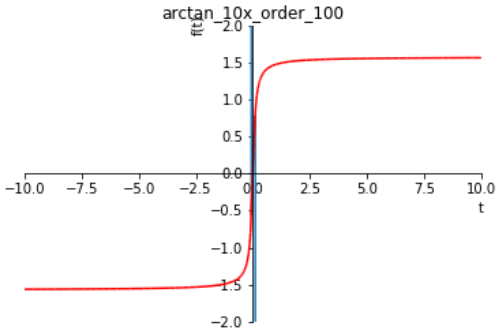
Kapitel 3

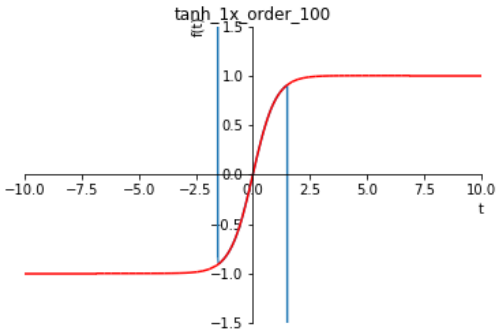
Notizen

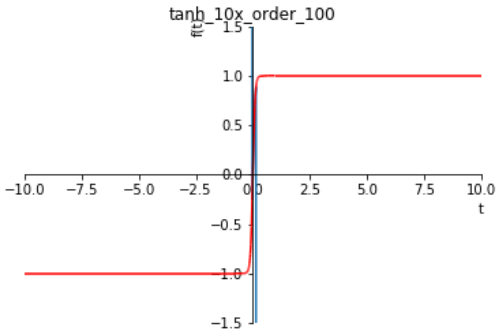
inhaltlich noch zu verarbeitende Punkte: identification with prior knowledge reibung rauschen vergleichsparameter: tspan, dt, u0, (lambda), multra, p? NN

3.1 UDE









3.1.1 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 Autflösung viel zu gering)
bei erhöhung der auflösung wird trainingszeit enorm groß NN approximiert den verlauf der ableitungen
sindy mal mit anderen anfangswerten trainieren

3.1.2 9.7.20

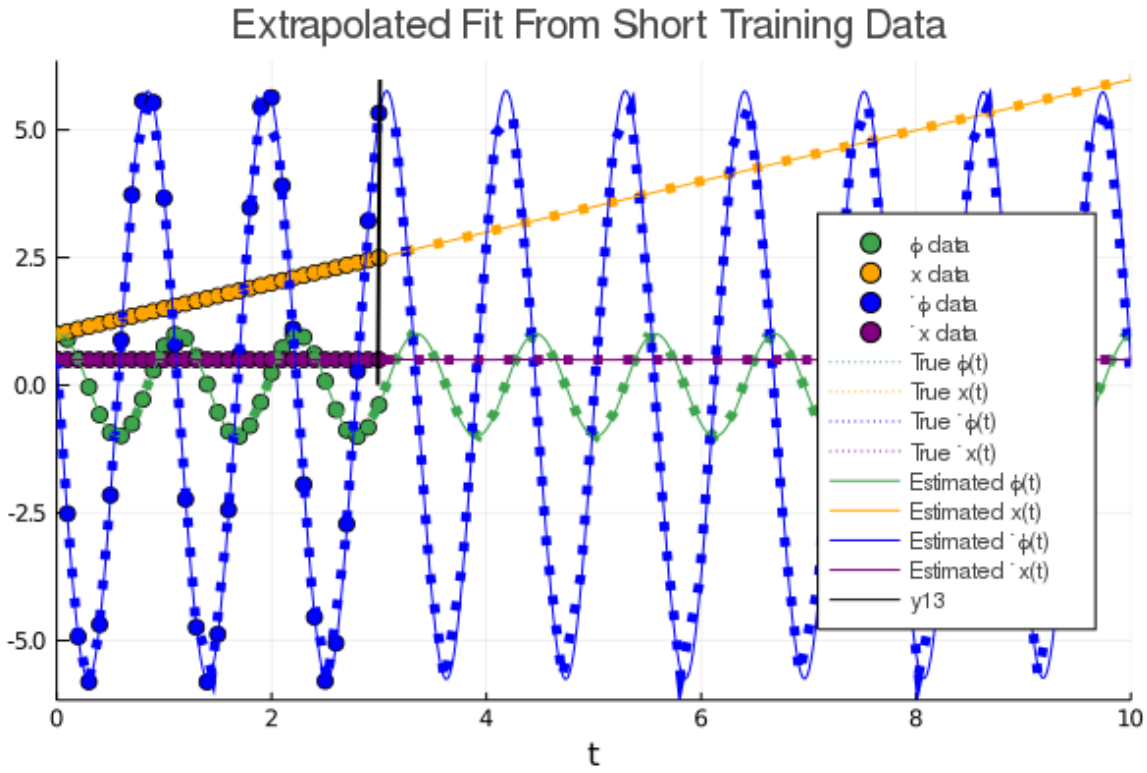
ude keine option für multiple trajectories / threshhold???

ude + sindy

definitorische Gleichungen bekannt
dt=0.1
sindy did not converge

$$du_1 = cos(u_3) * p_1 \tag{3.1}$$
$$du_2 = sin(u_3) * p_2 \tag{3.2}$$
$$du_3 = sin(u_1) * p_3 + p_4 * u_1 \tag{3.3}$$
$$du_4 = sin(u_3) * p_5 \tag{3.4}$$

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



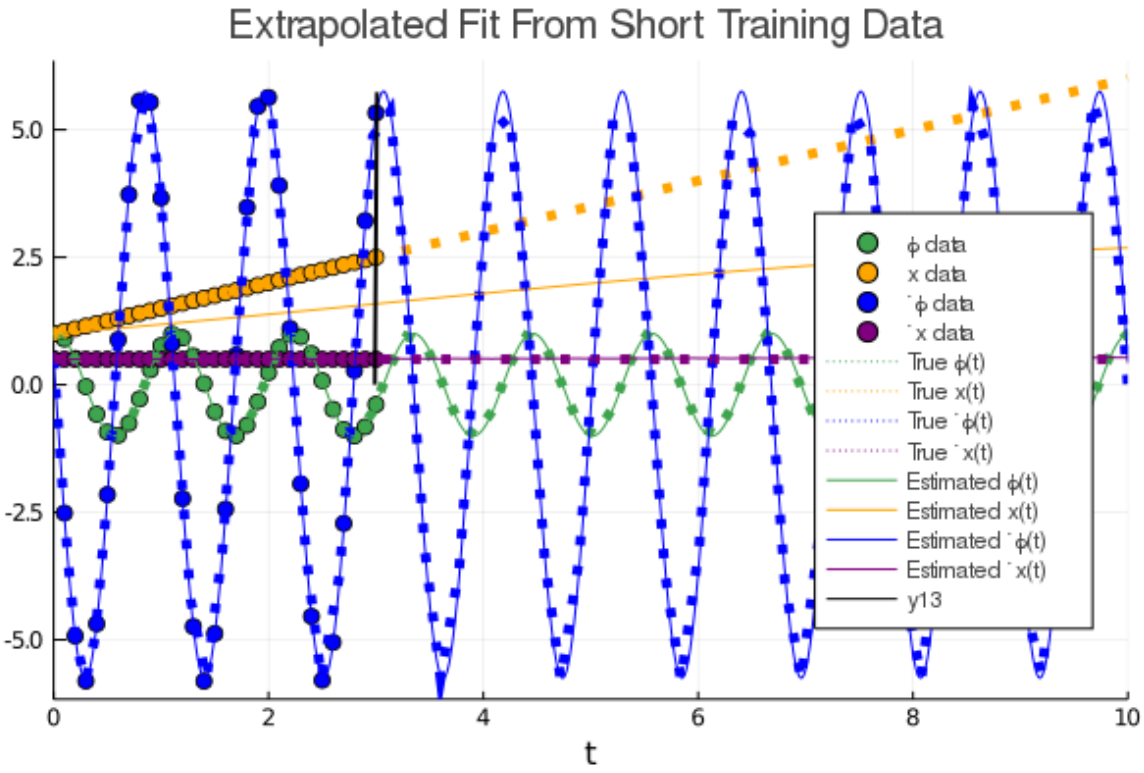
im Vergleich: nur den Sindy Part in PySindy ausgelagert ergibt: (dieser Vergleich ist nicht sinnvoll? pysindy kennt ja die def. gl nicht)

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

keine Gleichungen bekannt
dt=0.1
sindy did not converge

$$
\begin{aligned}
du_1 &= p_1 * u_3 && (3.5) \\
du_2 &= sin(u_2) * p_2 && (3.6) \\
du_3 &= sin(u_1) * p_3 + p_4 * u_1 && (3.7) \\
du_4 &= cos(u_1) * p_5 && (3.8)
\end{aligned}
$$

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



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

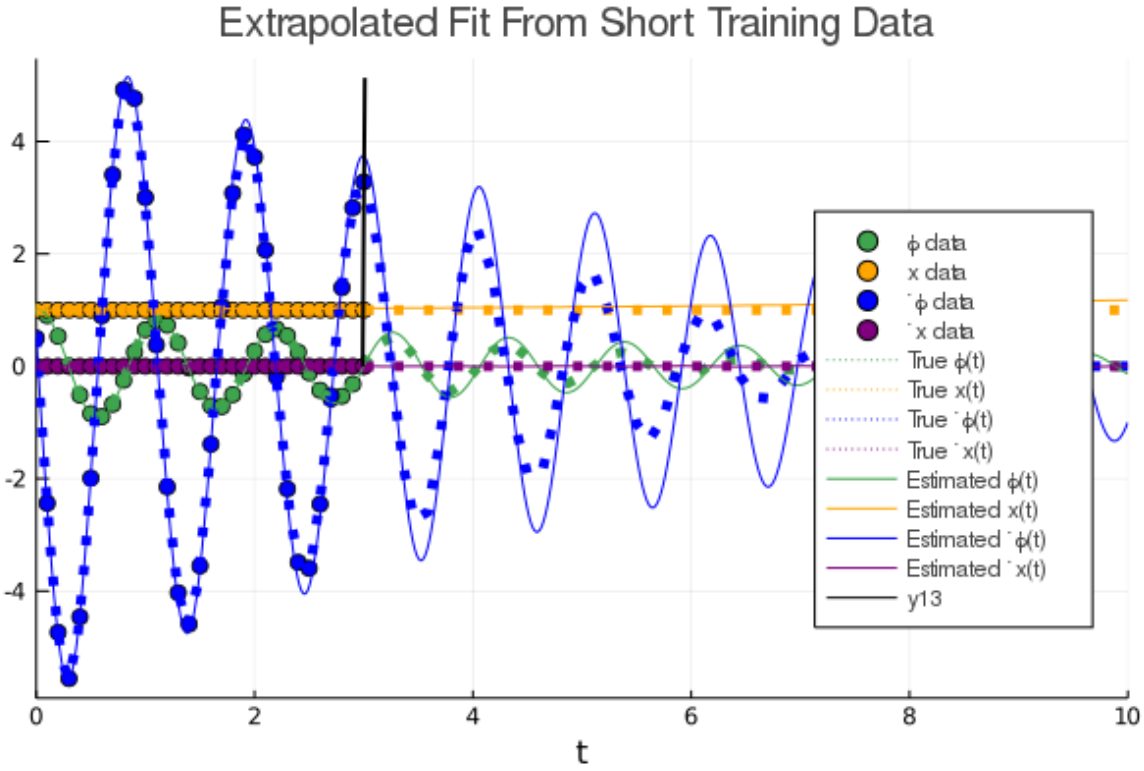
$$
\begin{aligned}
phi' &= 1.111phi + 0.648x + 1.002phidot + -503701656.178xdot + -1.228sin(phi) \\
&\quad + 562779966.638sin(xdot) + -0.050cos(phi) + 0.676cos(x) + -20465608.922cos(xdot) \\
x' &= 0.833phi + -0.031x + 450669011.982xdot + -0.967sin(phi) \\
&\quad + -503527393.951sin(xdot) + 0.157cos(phi)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

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]



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:

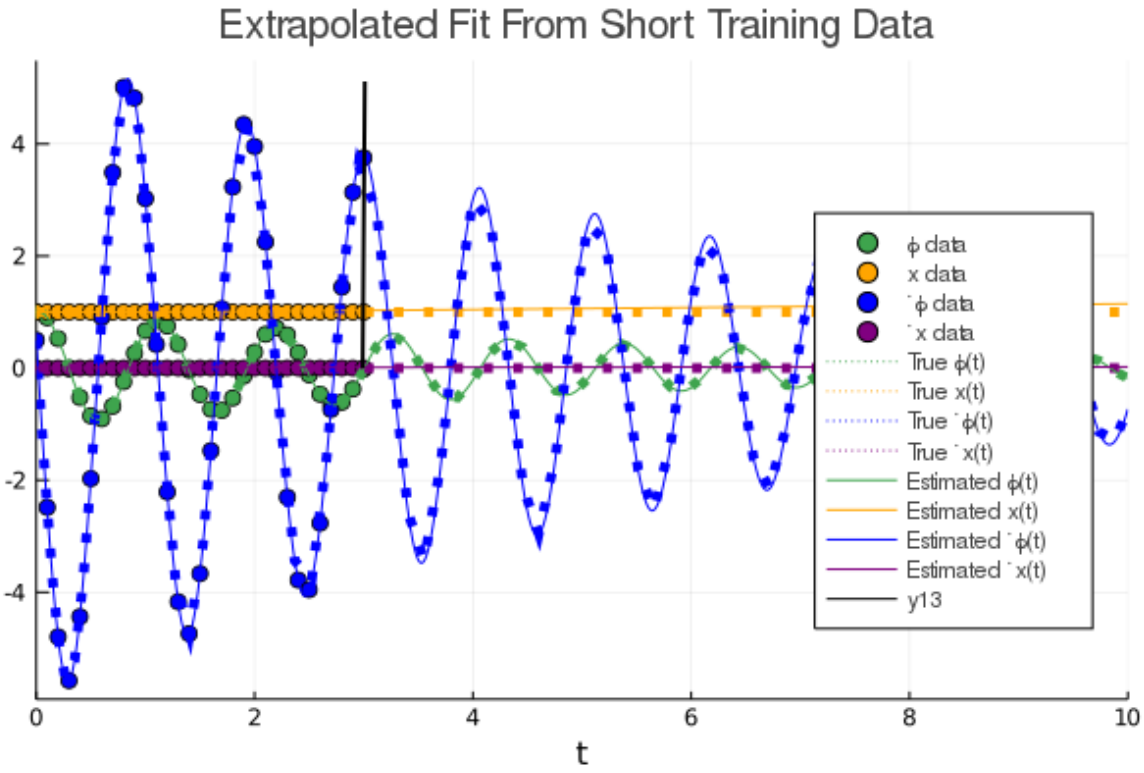


Abbildung 2 – ude fric viskos d1 03

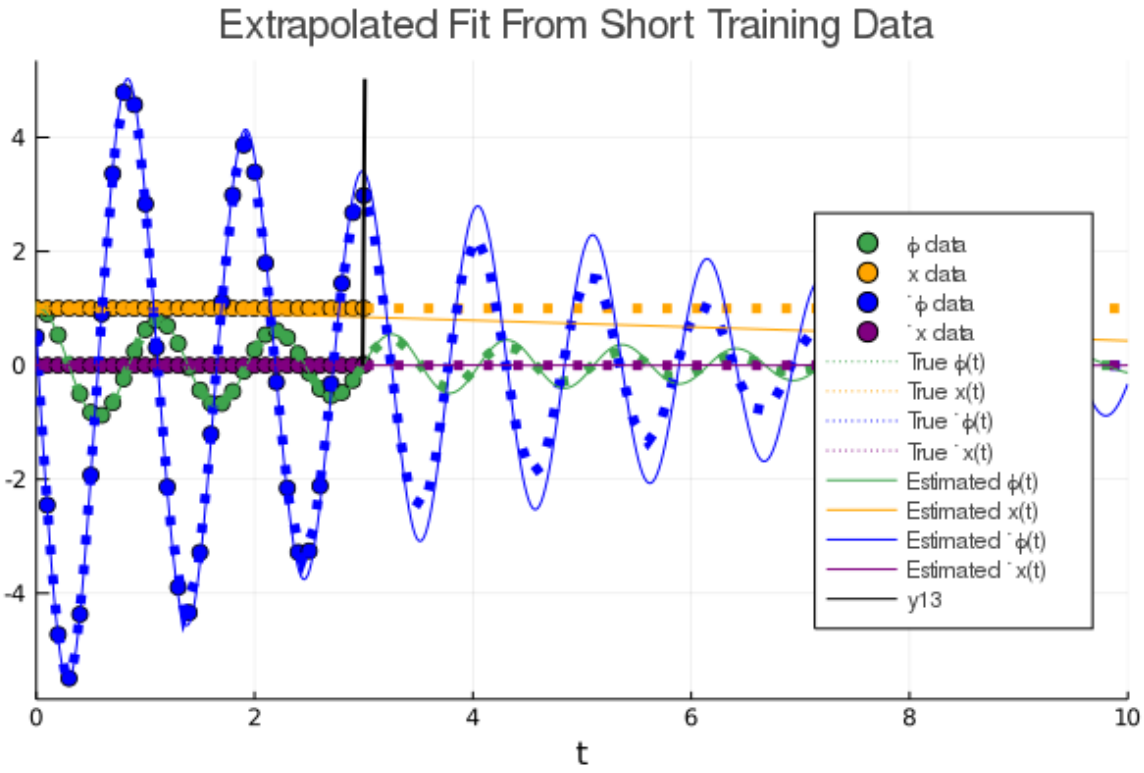


Abbildung 3 – ude fric viskos und haft d1 03 d2 05

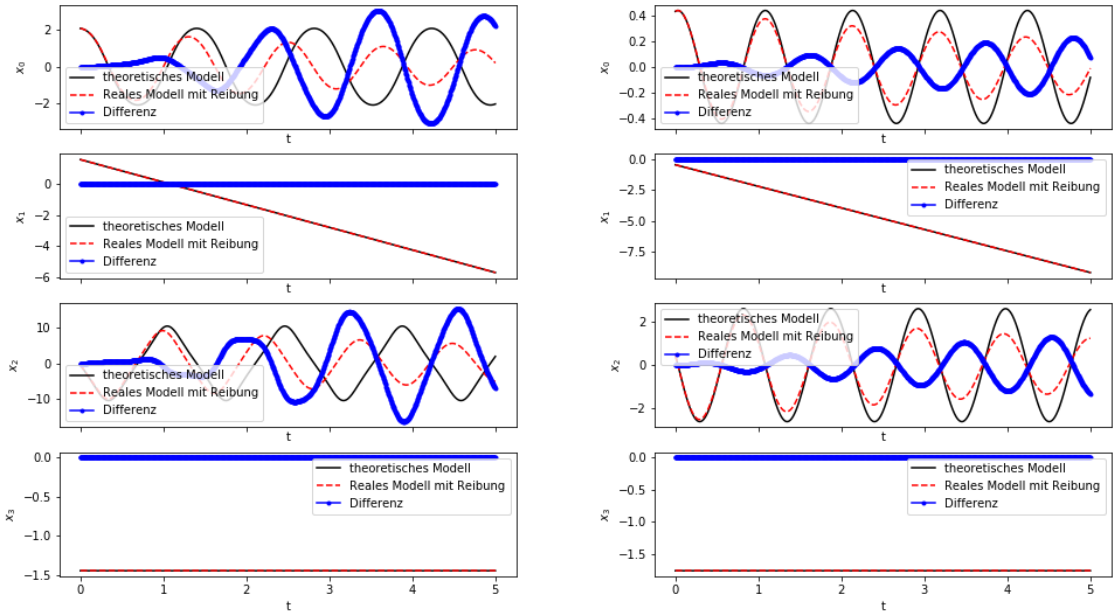


Abbildung 4 – nur viskose Reibung

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

3.2 Sindy

3.2.1 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 $x(t) = [x_1(t), x_2(t), ..., x_n(t)]$. Gesucht ist die Funktion der zeitlichen Ableitung des Zustandes $\frac{\mathrm{d}x(t)}{\mathrm{d}t} = f(x(t))$, 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

$$\frac{\mathrm{d}x}{\mathrm{d}t} = f(x) = \begin{bmatrix} f_1(x) \\ f_2(x) \end{bmatrix} = \begin{bmatrix} 1 + 2x_1 + x_1 x_2^2 \\ x_1^3 - 3x_2 \end{bmatrix} \tag{3.9}$$

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 $t_1, t_2, ..., t_m$). 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:

$$X = \begin{bmatrix} x_1(t_1) & x_2(t_1) & \dots & x_n(t_1) \\ x_1(t_2) & x_2(t_2) & \dots & x_n(t_2) \\ \vdots & \vdots & & \vdots \\ x_1(t_m) & x_2(t_m) & \dots & x_n(t_m) \end{bmatrix} \in \mathbb{R}^{m \times n}, \tag{3.10}$$

$$\dot{X} = \begin{bmatrix} \dot{x_1}(t_1) & \dot{x_2}(t_1) & \dots & \dot{x_n}(t_1) \\ \dot{x_1}(t_2) & \dot{x_2}(t_2) & \dots & \dot{x_n}(t_2) \\ \vdots & \vdots & & \vdots \\ \dot{x_1}(t_m) & \dot{x_2}(t_m) & \dots & \dot{x_n}(t_m) \end{bmatrix} \in \mathbb{R}^{m \times n}. \tag{3.11}$$

Nun muss man die Bibliothek Θ 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

$$\Theta(X) = \begin{bmatrix} | & | & & | \\ \theta_1(X) & \theta_2(X) & \dots & \theta_\ell(X) \\ | & | & & | \end{bmatrix} \in \mathbb{R}^{m \times L}. \tag{3.12}$$

Wählt man beispielsweise für θ_1 die Sinusfunktion und für θ_2 Monome zweiten Grades so ergeben sich

$$\theta_1(X) = \begin{bmatrix} | & | & & | \\ \sin(x_1(t)) & \sin(x_2(t)) & \dots & \sin(x_n(t)) \\ | & | & & | \end{bmatrix}, \tag{3.13}$$

$$\theta_2(X) = \begin{bmatrix} | & | & & | & | & & | \\ x_1(t)^2 & x_1(t)x_2(t) & \dots & x_2(t)^2 & x_2(t)x_3(t) & \dots & x_n^2(t) \\ | & | & & | & | & & | \end{bmatrix}, \tag{3.14}$$

wobei die Rechenoperationen alle elementweise zu lesen sind.
Gesucht sind nun Linearkombinationen von Bibliotheksfunktionen, sodass gilt

$$f_i(x) = \Theta(x^T)\xi_i. \tag{3.15}$$

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

$$\dot{X} \approx \Theta(X)\Xi. \tag{3.16}$$

[2] 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

$$\dot{X} \approx \Theta(X)\Xi \tag{3.17}$$

Intuitively, I would solve this by using the pseudo-inverese

$$\Xi = (\Theta(X))^{+} \dot{X}. \tag{3.18}$$

With $\Theta(X)$ having full collumn rank, we get

$$\Xi = \left(\Theta(X)^T\Theta(X)\right)^{-1} \Theta(X)^T\dot{X}. \tag{3.19}$$

But in the code implemented is the following equation:

$$\Xi_{\text{new}} = \left(\Theta(X)^T\Theta(X) + I\right)^{-1} \left(\Theta(X)^T\dot{X} + \Xi_{\text{old}}\right) \tag{3.20}$$

with the first coefficient matrix being

$$\Xi_{\text{old, 0}} = \left(\Theta^T\Theta\right)^{-1} \Theta^T\dot{X}. \tag{3.21}$$

What is the reasoning behind using 3.20 over the more intuitive 3.19? (I do understand the motivation behind repeating the calculation and successively eliminating small coefficients, but why do you use this formula?)

Additionally I noted that using

$$\Theta^T\dot{X} \approx \Theta^T\Theta\Xi_{\text{old}} \tag{3.22}$$

we can simplify 3.20

$$\begin{aligned} \Xi_{\text{new}} &= \left(\Theta^T\Theta + I\right)^{-1} \left(\Theta^T\Theta\Xi_{\text{old}} + \Xi_{\text{old}}\right) \\ &\approx \left(\Theta^T\Theta + I\right)^{-1} \left(\Theta^T\Theta + I\right) \Xi_{\text{old}} \\ &\approx \Xi_{\text{old}} \end{aligned} \tag{3.23}$$

which seems odd to me.

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 ëxactßolution of the equation.

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

System: Volterra DGL

$$\dot{x} = \alpha x + \beta xy \tag{3.24}$$
$$\dot{y} = \gamma y + \delta xy \tag{3.25}$$

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
α	1.3	1.291	1.300
β	-0.9	-0.850	-0.900
γ	-1.8	-1.801	-1.800
δ	0.8	0.753	0.800

Fall 2: Zentraldifferenz für Ableitungen verwendet

Parameter	Nominal	DiffEq	PySindy
α	1.3	1.301	1.302
β	-0.9	-0.683	-0.903
γ	-1.8	-1.816	-1.802
δ	0.8	0.614	0.798

Fall 3: Ableitungen durch Neuronales Netz gelernt, gleiche Auflösung

Parameter	Nominal	DiffEq	PySindy
α	1.3	1.438	1.289
β	-0.9	-0.226	-0.868
γ	-1.8	-1.814	-1.780
δ	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
α	1.3	1.471	1.294
β	-0.9	-0.353	-0.887
γ	-1.8	-1.818	-1.786
δ	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
α	1.3	1.477	1.294
β	-0.9	-0.367	-0.889
γ	-1.8	-1.820	-1.786
δ	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?
Ξ_{alt} aus PySindy zu streichen macht den algorithmus kaputt

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 zb roessler
vgl bei "high res"von X aus NN vs X aus odesolver
problem bei verkettung: sin(2x), parameter innerhalb von funktionen nicht in library darstellbar
bei wp: pysindy kann mittels choslesky zerlegung nicht reguläre matrix invertieren???
bei wp: durch mehrere 0 spalten in library pseudoinverse nicht mehr positiv definit,

behoben durch +I, allerdings dann ergebnisse schlechter
test17

3.3 Wagen-Pendel

3.3.1 mit Reibung

$$\dot{\vec{x}} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} \varphi \\ x \\ \dot{\varphi} \\ \dot{x} \end{pmatrix} = f(\vec{x}) + g(\vec{x})u \tag{3.26}$$

$$= \begin{pmatrix} \dot{\varphi} \\ \dot{x} \\ -\frac{(m_1+m_2)R(\dot{\varphi})+m_2^2 s_2^2 \dot{\varphi}^2 \sin\varphi\cos\varphi+(m_1+m_2)gm_2 s_2 \sin\varphi}{m_2 s_2^2(m_1+m_2\sin^2\varphi)} \\ \frac{R(\dot{\varphi})\cos\varphi+gm_2 s_2 \sin\varphi\cos\varphi+m_2 s_2^2\dot{\varphi}^2\sin\varphi}{s_2(m_1+m_2\sin^2\varphi)} \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ -\frac{m_2 s_2\cos\varphi}{m_2 s_2^2(m_1+m_2\sin^2\varphi)} \\ \frac{s_2}{s_2(m_1+m_2\sin^2\varphi)} \end{pmatrix} F \tag{3.27}$$

3.4 errors

Ude:
$AssertionError : length(b) == length(variables(b))inunknown_sys :$
wenn λ so gewählt, dass ganze Zeilen rausfallen
nützliche Befehle:

Literatur

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

[2] Brian M. de Silva u. a. “PySINDy: A Python package for the Sparse Identification of Nonlinear Dynamics from Data”. In: (17. Apr. 2020). arXiv: 2004.08424 [math.DS].