

Praktikum MATLAB®/Simulink® II

Prof. Dr.-Ing. U. Konigorski
Musterlösung



TECHNISCHE
UNIVERSITÄT
DARMSTADT

REGELUNGSTECHNIK **rtm**
UND MECHATRONIK

Praktikum MATLAB®/Simulink® II

Prof. Dr.-Ing. U. Konigorski

Musterlösung



Technische Universität Darmstadt
Institut für Automatisierungstechnik und Mechatronik
Fachgebiet Regelungstechnik und Mechatronik
Prof. Dr.-Ing. U. Konigorski

Landgraf-Georg-Straße 4
64283 Darmstadt
Telefon 06151/16-25200
www.rtm.tu-darmstadt.de

Das Gesamtdokument ist unter CC BY-ND veröffentlicht:



<https://creativecommons.org/licenses/by-nd/4.0/>

Der Inhalt dieses Dokuments ausschließlich der Logos, des Layouts und der Schriftarten ist unter CC BY-SA veröffentlicht:



<https://creativecommons.org/licenses/by-sa/4.0/>



Inhaltsverzeichnis

1	Modellierung und Simulation eines Schlitten-Pendel-Systems	5
2	Steuerbarkeit und Beobachtbarkeit	23
3	LQ-Regelung und Animation	39
4	Beobachterentwurf – Benutzeroberflächen	65
5	Aufschwingsteuerung für das Pendel	89
6	Trajektorienfolgeregelung	111



Versuch 1

Modellierung und Simulation eines Schlitten-Pendel-Systems

1 Versuchsdurchführung	6
1.1 Aufgaben	6
1.2 WICHTIG: Hinweis zur Erstellung des Versuchsberichtes	21

1 Versuchsdurchführung

Die folgenden Aufgaben sind teilweise während des Versuchs zu lösen und am nächsten Versuchsnachmittag oder spätestens eine Woche später mit dem Protokoll den Betreuern abzugeben. Die Aufgaben, die während des Versuchs zu lösen sind, sind handschriftlich in den umrandeten Feldern einzutragen.

1.1 Aufgaben

Legen Sie die Parameter des Pendels und ihre Zahlenwerte in einem Skript (`initModell.m`) ab, so dass diese geändert werden können, ohne die Simulink-Modelle und Funktionen anpassen zu müssen.

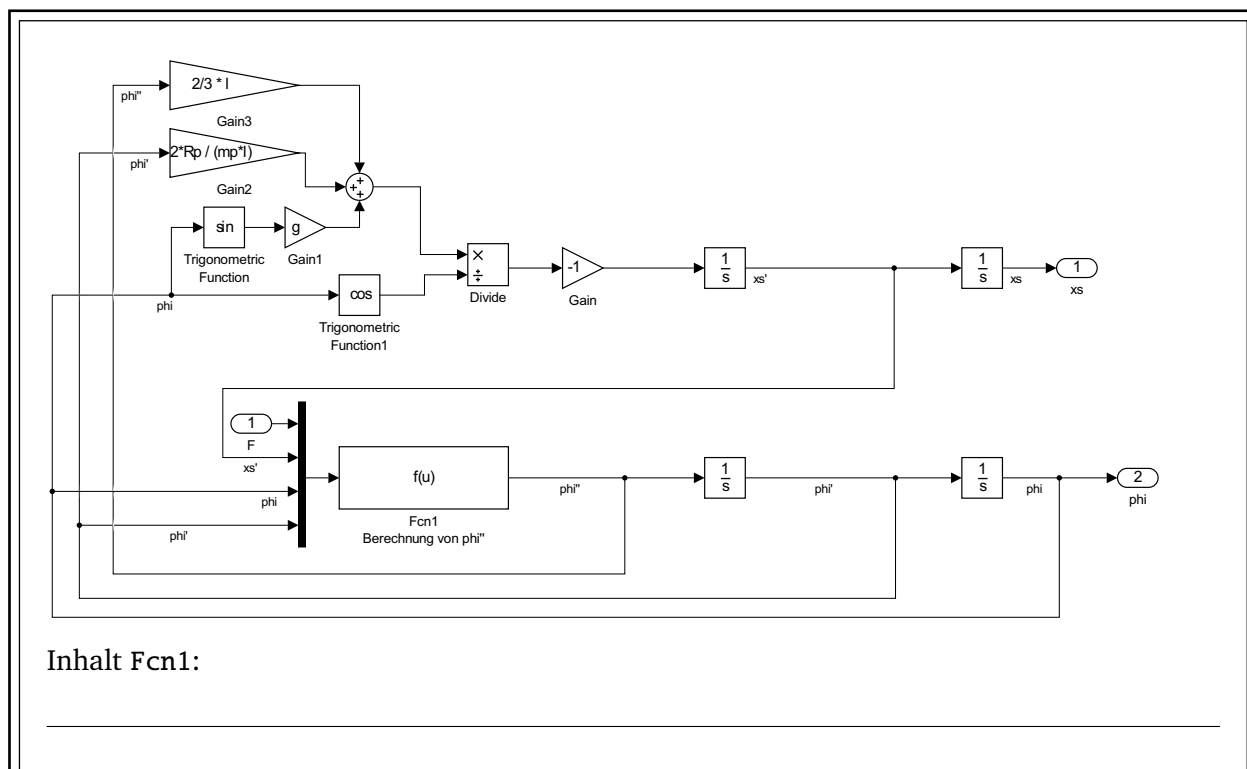
Aufgabe 1.3 (Durchführung/Nachbearbeitung):

Das **nichtlineare** Modell soll jeweils mit Hilfe von

- elementaren Funktionsblöcken der Math-Operations-Library und Fcn-Blocks,
- MATLAB-Function-Blocks sowie
- einer M-File S-Function

realisiert werden. Bearbeiten Sie die folgenden Aufgaben und implementieren Sie die Modelle anschließend in Simulink.

- Ergänzen Sie die Blöcke zur Berechnung der Position x_s und des Winkels φ und geben Sie die Funktion des Blocks Fcn1 an!

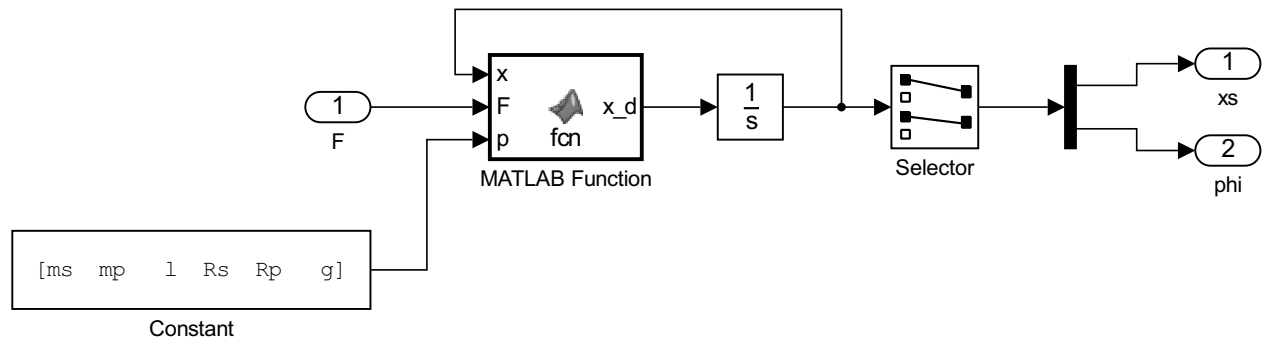



```

1      ((ms + mp) * (g * sin(u(3))) + 2 * Rp/(mp*l) * u(4)) + mp * u(4)^2 * 1/4 * sin(2*u(3)) + cos(u(3)) * (u(1) - Rs * u(2))) / (mp * 1/2 * cos(u(3))^2 - (ms + mp) * 2/3 * 1)

```

- Wie lautet der Inhalt des MATLAB-Function-Blocks MATLAB Function? Wie müssen Sie den Anfangswert des Integrators wählen, damit der Pendelstab zu Beginn nach oben zeigt?



```

function x_d = fcn(x, F, p)
%#codegen

%xs = x(1);
xs_d = x(2);
phi = x(3);
phi_d = x(4);

ms = p(1);
mp = p(2);
l = p(3);
Rs = p(4);
Rp = p(5);
g = p(6);

phi_dd = ( ...
    (ms + mp) * (g * sin(phi) + 2 * Rp/(mp*l) * phi_d) + ...
    mp * phi_d^2 * 1/4 * sin(2*phi) + ...
    cos(phi) * (F - Rs * xs_d) ...
    ) / (mp * 1/2 * cos(phi)^2 - (ms + mp) * 2/3 * l);

xs_dd = -1/cos(phi) * ...
    (g * sin(phi) + 2 * Rp/(mp * l) * phi_d + 2/3 * l * phi_dd);

x_d = [xs_d; xs_dd; phi_d; phi_dd];

```

Der Anfangswert des Integrators muss auf $[0; 0; \pi; 0]$ gesetzt werden.

- Vervollständigen Sie das M-File an die vorliegende Aufgabenstellung an den entsprechenden Stellen.

Hinweis: Bei den Praktikumsunterlagen ist eine Vorlage für die S-Function zur Verfügung gestellt (siehe auch den Anhang zum Versuch 1 im Skript).

```
function sFunModell(block)

    setup(block);

end

% *****
% Initialisierung
% *****
function setup(block)

    % Anzahl der Ein-/Ausgänge
    block.NumInputPorts = 1;
    block.NumOutputPorts = 2;

    % Eigenschaften des Eingangs
    block.InputPort(1).Dimensions = 1;
    block.InputPort(1).DatatypeID = 0; % double
    block.InputPort(1).Complexity = 'Real';
    block.InputPort(1).DirectFeedthrough = false;
    block.InputPort(1).SamplingMode = 'Sample';

    % Eigenschaften des 1. Ausgangs
    block.OutputPort(1).Dimensions = 1;
    block.OutputPort(1).DatatypeID = 0; % double
    block.OutputPort(1).Complexity = 'Real';
    block.OutputPort(1).SamplingMode = 'Sample';

    % Eigenschaften des 2. Ausgangs
    block.OutputPort(2).Dimensions = 1;
    block.OutputPort(2).DatatypeID = 0; % double
    block.OutputPort(2).Complexity = 'Real';
    block.OutputPort(2).SamplingMode = 'Sample';

    % Anzahl der Zustände
    block.NumContStates = 4;

    % Anzahl der Parameter
    block.NumDialogPrms = 7;

    % Abtastzeit definieren -> zeitkontinuierlich
    block.SampleTimes = [0 0];

    % weitere Methoden registrieren
    block.RegBlockMethod('InitializeConditions', @InitializeConditions);
    block.RegBlockMethod('Outputs', @Outputs);
    block.RegBlockMethod('Derivatives', @Derivatives);
```

```

        block.RegBlockMethod('Terminate', @Terminate);

end

% *****
% Anfangsbedingungen setzen
% *****
function InitializeConditions(block)

    phi0 = block.DialogPrm(7).Data;
    block.ContStates.Data = [0; 0; phi0; 0];

end

% *****
% Ausgänge berechnen
% *****
function Outputs(block)

    % Zustände auslesen
    x      = block.ContStates.Data;
    xs     = x(1);
    phi    = x(3);

    block.OutputPort(1).Data = xs;
    block.OutputPort(2).Data = phi;

end

% *****
% Ableitungen berechnen
% *****
function Derivatives(block)

    % Parameter auslesen
    ms = block.DialogPrm(1).Data;
    Rs = block.DialogPrm(2).Data;
    mp = block.DialogPrm(3).Data;
    Rp = block.DialogPrm(4).Data;
    l  = block.DialogPrm(5).Data;
    g  = block.DialogPrm(6).Data;

    % Zustände auslesen
    x      = block.ContStates.Data;
    xs     = x(1);
    xs_d   = x(2);
    phi    = x(3);
    phi_d  = x(4);

    % Eingang auslesen
    F = block.InputPort(1).Data(1);

    % Ableitungen berechnen
    phi_dd = ( ...

```

```

        (ms + mp) * (g * sin(phi) + 2 * Rp/(mp*l) * phi_d) + ...
        mp * phi_d^2 * l/4 * sin(2*phi) + ...
        cos(phi) * (F - Rs * xs_d) ...
    ) / (mp * l/2 * cos(phi)^2 - (ms + mp) * 2/3 * l);

xs_dd = -1/cos(phi) * ...
        (g * sin(phi) + 2 * Rp/(mp * l) * phi_d + 2/3 * l * phi_dd);

% Ableitungen zuweisen
block.Derivatives.Data = [xs_d; xs_dd; phi_d; phi_dd ];

end

% *****
% Aufräumen (wenn nötig)
% *****
function Terminate(block)
end

```

- Welche Vor- und Nachteile bieten die unterschiedlichen Möglichkeiten der Realisierung nichtlinearer Modelle?

	Vorteile	Nachteile
Math-Operations Library	<ul style="list-style-type: none"> – Darstellung des Systems in Form eines Blockschaltbildes – leichter Zugriff auf Zwischenergebnisse 	<ul style="list-style-type: none"> – unübersichtliche Darstellung von komplexen Funktionen
Fcn-Block	<ul style="list-style-type: none"> – kompakte und übersichtliche Darstellung von linearen und nichtlinearen Funktionen 	<ul style="list-style-type: none"> – keine Durchführung von Matrixoperationen möglich – kein Zugriff auf Zwischenergebnisse möglich
MATLAB-Function-Block	<ul style="list-style-type: none"> – übersichtliche Realisierung von sehr komplexen Funktionen – einfache Implementierung – Debuggen möglich 	<ul style="list-style-type: none"> – Kompilieren des Modells vor dem Ausführen benötigt etwas Zeit
M-File S-Function	<ul style="list-style-type: none"> – übersichtliche Realisierung von sehr komplexen Funktionen – Debuggen möglich – zusätzlich zu der MATLAB-Programmiersprache M auch die Verwendung der Programmiersprachen C, C++, Fortran und Ada möglich 	<ul style="list-style-type: none"> – komplexe Implementierung – Zugriff auf Zwischenergebnisse aufwändig

Aufgabe 1.4 (Durchführung/Nachbearbeitung):

Das **lineare** Modell soll als Zustandsraummodell implementiert werden. Bearbeiten Sie die Aufgabe und implementieren Sie das Modell anschließend in Simulink. Der Block *Integrator* und die Bibliotheken *Sinks* und *Sources* können dabei ohne Einschränkung verwendet werden.

- Vervollständigen Sie die Funktion `initLinear`, welche die Matrizen **A**, **B**, **C** und **D** für das ZRM in Abhängigkeit des Arbeitspunktes initialisiert.

```
function [A, B, C, D] = initLinear( l, mp, ms, Rs, Rp, g, AP )
% function [A, B, C, D] = initLinear( lPendel, mPendel, mSchlitten, RSchlitten→
    ←, RPendel, g, arbeitspunkt)
% Arbeitspunkt:
% arbeitspunkt = 0; oder
% arbeitspunkt = pi;
%
% Zustände:
% x = [ xs, xs_d, phi, phi_d ]'
%

if ( AP == 0 )
    % Hängendes Pendel
    A = [
        [0, 1, 0, 0];
        [0, -4*Rs, 3*g*mp, 6*Rp/l] / (4*ms+mp);
        [0, 0, 0, 1];
        [0, 6*Rs, -6*g*(ms+mp), -12*Rp*(ms+mp)/l/mp] / (1*(4*ms+mp));
    ];

    B = [0; 4; 0; -6/l]/(4*ms+mp);

    C = [ 1, 0, 0, 0;
          0, 0, 1, 0 ];

elseif ( AP == pi )
    % Stehendes Pendel
    A = [
        [0, 1, 0, 0];
        [0, -4*Rs, 3*g*mp, -6*Rp/l] / (4*ms+mp);
        [0, 0, 0, 1];
        [0, -6*Rs, 6*g*(ms+mp), -12*Rp*(ms+mp)/l/mp] / (1*(4*ms+mp));
    ];

    B = [0; 4; 0; 6/l]/(4*ms+mp);

    C = [ 1, 0, 0, 0;
          0, 0, 1, 0 ];

else
    % Andere Werte für AP entweder nicht sinnvoll, da kein
    % AP oder schlicht Vielfache von pi und daher nicht relevant.
    error( ['Wert_arbeitspunkt_nicht_unterstützt.'] );
end

D = [0; 0];
```

end

Die nachfolgenden Aufgaben sind in Ihrem Protokoll zu beantworten.

Verwenden Sie für die nachfolgenden Simulationen den *ode45* (Dormand-Prince)-Solver und eine geeignete maximale Schrittweite (auf die MATLAB-Ausgabe während der Simulation achten). Verringern Sie die maximale Schrittweite, wenn die Graphen sehr kantig wirken.

Aufgabe 1.5 (Durchführung/Nachbearbeitung):

Untersuchen Sie das Verhalten bzw. die Reaktion der erstellten Modelle anhand der Auswertung der Signale $x_s(t)$ und $\varphi(t)$ auf eine sinusförmige Anregung mit

$$F(t) = \sin(\omega \cdot t) \quad \text{und} \quad \omega = 1 \text{ rad/s} \quad \text{für die Zeitspanne} \quad t = 0 \text{ s} \quad \text{bis} \quad t = 20 \text{ s}.$$

- Sind die Simulationsergebnisse aller drei nichtlinearen Modelle identisch ($\varphi(t=0) = 0 \text{ rad}$)? Fertigen Sie dazu folgenden Plot mit Hilfe von `subplot` und `linkaxes` an: Stellen Sie im ersten Subplot die anregende Kraft, im zweiten die drei absoluten Positionen des Schlittens und im dritten die absoluten Winkel dar. Vergessen Sie nicht die sinnvolle Achsenskalierung, die Achsenbeschriftung und die Verwendung unterschiedlicher Strichtypen!

(Für die Erstellung der Plots beachten Sie bitte Abschnitt Physikalische Modellbildung → Simulation → Darstellung der Ergebnisse im Skript von Versuch 1. Die Nichtbeachtung der Richtlinien führt zu Punktabzug!)

Die Simulationsergebnisse sind in Abbildung Lösung 1 dargestellt. Die Simulationen liegen exakt übereinander und sind daher nicht voneinander zu unterscheiden.

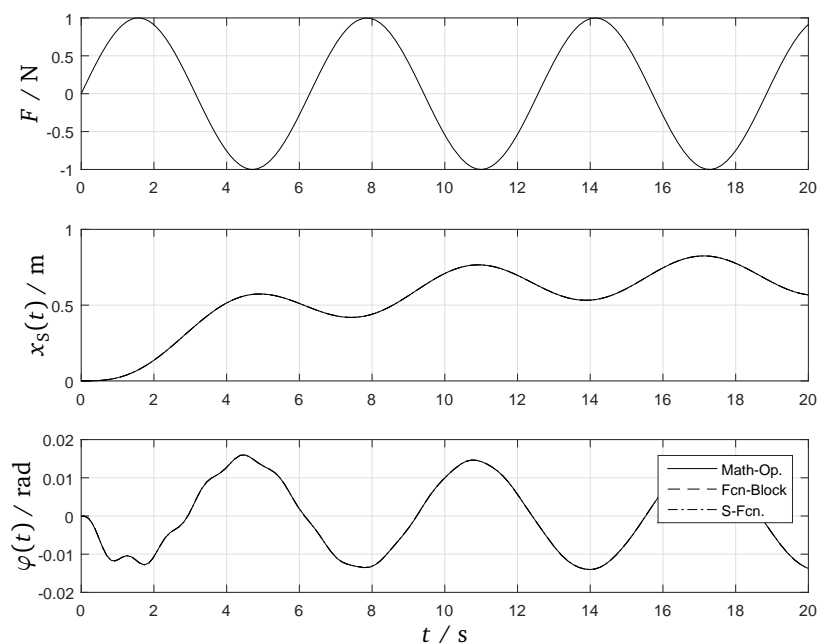


Abbildung Lösung 1: Vergleich der nichtlinearen Modelle

Hinweis:

Suchen Sie für die folgenden Aufgaben ein geeignetes nichtlineares Modell zum Vergleich mit dem linearisierten Modell aus.

- Stellen Sie die Simulationsergebnisse von $F(t)$, $x_s(t)$ und $\varphi(t)$ des linearisierten und eines nicht-linearen Modells (Absolutwerte angeben! Siehe Hinweis im Abschnitt Simulation im Skript zum Versuch 1.) jeweils für $\varphi(t=0) = 0\text{rad}$ und $\varphi(t=0) = \pi\text{rad}$ dar. Passen Sie die Anfangsbedingungen dem Arbeitspunkt an.

Da alle nichtlinearen Modelle gleich sind, ist es egal, welches Modell mit dem linearisierten Modell verglichen wird. Hier wird die Modellierung mittels S-Function herangezogen. Abbildung Lösung 2 zeigt das linearisierte und das nichtlineare Modell am unteren Arbeitspunkt $\Phi_0 = 0\text{rad}$. Die Simulationen stimmen überein, für kleine Auslenkungen kann am unteren AP auch das lineare Modell verwendet werden. Die Linearisierung wurde korrekt durchgeführt.

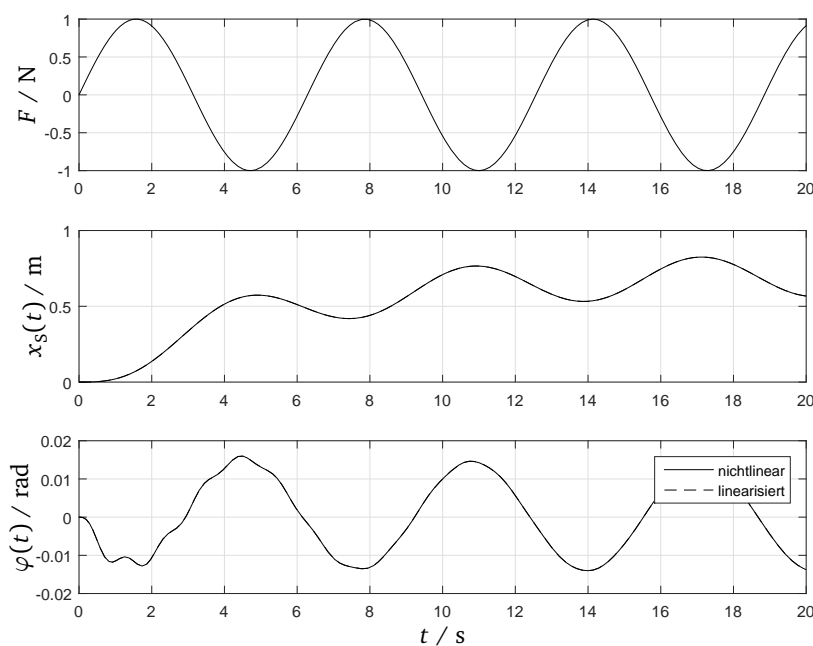


Abbildung Lösung 2: Vergleich des um $\Phi_0 = 0\text{rad}$ linearisierten und des nichtlinearen Modells

Abbildung Lösung 3 zeigt das linearisierte und das nichtlineare Modell am oberen Arbeitspunkt $\Phi_0 = \pi\text{rad}$. Das lineare Modell ist für kleine Auslenkungen gültig. In der Simulation des nichtlinearen Modells ist zu erkennen, dass der Pendelstab in die untere Position fällt und dort verbleibt.

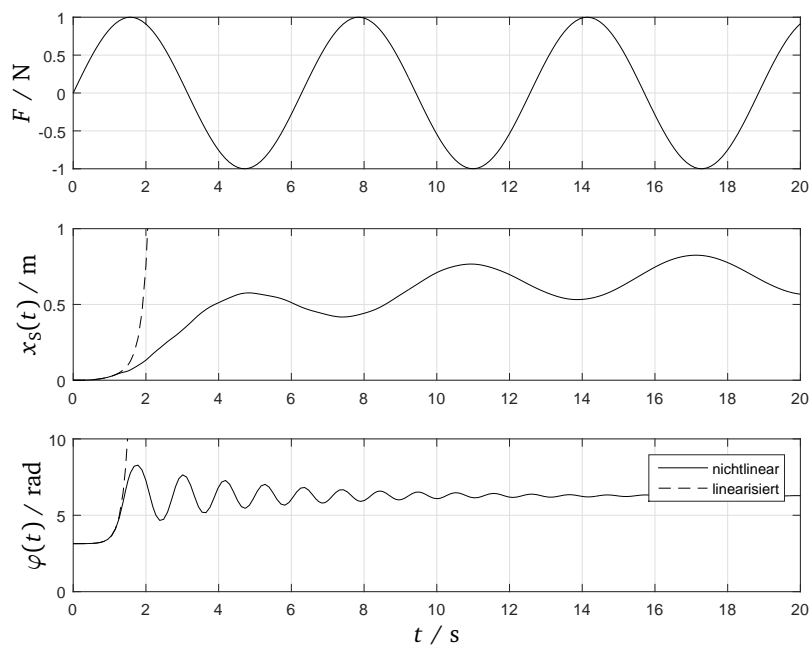


Abbildung Lösung 3: Vergleich des um $\Phi_0 = \pi \text{ rad}$ linearisierten und des nichtlinearen Modells

- Ist das Verhalten des Pendels (bzw. des Winkels $\varphi(t)$) plausibel? Beschreiben Sie dazu die erwartete und die simulierte Reaktion des Modells für die ersten zwei Sekunden der Simulation. (Zu beachten sind $F(t)$, $x_s(t)$ und $\varphi(t)$.)

Abbildung Lösung 4 zeigt einen vergrößerten Ausschnitt bei Linearisierung um den oberen Arbeitspunkt. Deutlich ist hier nochmals zu erkennen, dass die beiden Modelle für kleine Auslenkungen identisch sind.

Es wird zunächst eine positive Kraft auf den Schlitten ausgeübt. Damit bewegt sich der Schlitten wie zu erwarten nach rechts. (x_s nimmt größere (positive) Werte an.) Damit wird auch der untere Punkt des Pendels nach rechts bewegt. Da der Schwerpunkt des Pendels zunächst noch in der Ausgangslage verharret, ergibt sich damit eine Drehung des Pendels gegen den Uhrzeigersinn, d. h. in die positive Zählrichtung von φ . Dies ist auch in den Graphen zu erkennen.

Während das Pendel nach links herunterfällt, beschleunigt die Gewichtskraft zunächst diese Bewegung. Wenn das Pendel die untere Lage, d. h. $\varphi = 2\pi$ überschreitet, wirkt die Gewichtskraft dann jedoch bremsend, und es ist zu erwarten, dass sich eine (gedämpfte) Schwingung um diese untere Lage einstellt. Das nichtlineare System verhält sich in der Simulation auch entsprechend. Für das linearisierte System bedeutet jedoch jede weitere Zunahme des Winkels φ eine Zunahme der Winkelbeschleunigung des Pendelstabes, und damit aufgrund der Wechselwirkung zwischen Pendel und Schlitten auch indirekt eine Zunahme der Beschleunigung des Schlittens nach rechts. Diese Effekte lassen sich in den Plots ebenfalls erkennen.

Das Verhalten der Modelle ist damit plausibel.

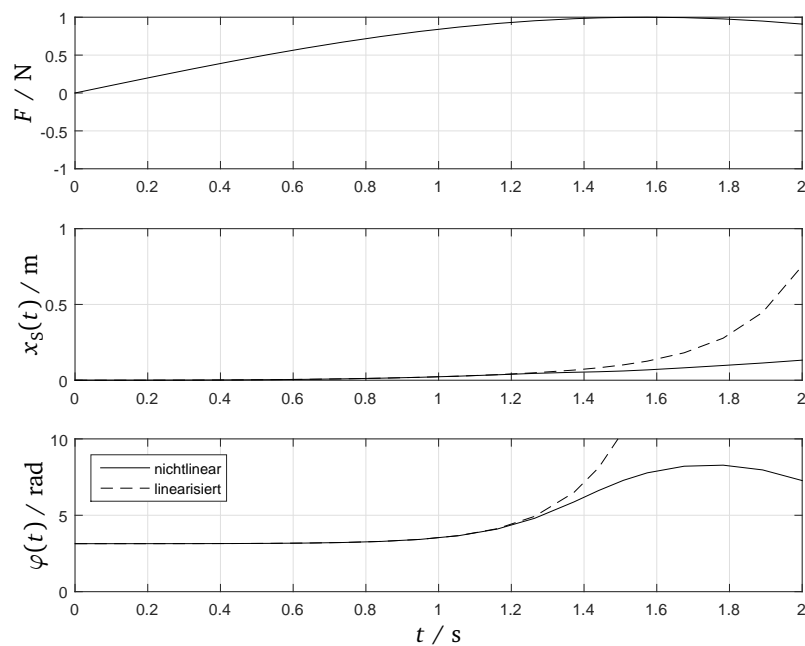


Abbildung Lösung 4: Vergleich des um $\Phi_0 = \pi \text{ rad}$ linearisierten und des nichtlinearen Modells

Hinweise:

Beachten Sie bei der Simulation, dass *Arbeitspunkt* und *Anfangsbedingungen* korrekt gewählt sind. Vergewissern Sie sich, dass der Arbeitspunkt und die Anfangsbedingungen des linearen Modells zu den Anfangsbedingungen des nichtlinearen Modells passen.

Um die einzelnen Größen miteinander vergleichen zu können, sollen diese entsprechend in gleiche Diagramme gezeichnet werden. Achten Sie darauf, wenn Sie mehrere Graphen in ein Diagramm zeichnen, dass diese auch im Schwarz-Weiß-Druck unterscheidbar dargestellt sind (verschiedene Marker/Strich-typen verwenden).

- Lässt sich anhand der Grafiken feststellen, ob die Linearisierung richtig durchgeführt wurde? Vergrößern Sie den für diese Aussage relevanten Bereich. Beschreiben Sie das Verhalten des linearisierten Modells.

Da die Verläufe der Schlittenposition und des Winkels im Bereich von kleinen Auslenkungen um den gewählten Arbeitspunkt deckungsgleich sind, ist anzunehmen, dass die Linearisierung richtig durchgeführt wurde. Ein vergrößerter Ausschnitt ist in Abbildung Lösung 4 dargestellt.

Aufgabe 1.6 (Durchführung/Nachbearbeitung):

Nun ist eine rechteckförmige Anregung gemäß Abbildung 1.1 anzusetzen.

- Vergleichen Sie das Verhalten des linearisierten und eines nichtlinearen Modells. Betrachten Sie auch hier beide Arbeitspunkte und erstellen Sie zu beiden Arbeitspunkten Grafiken.

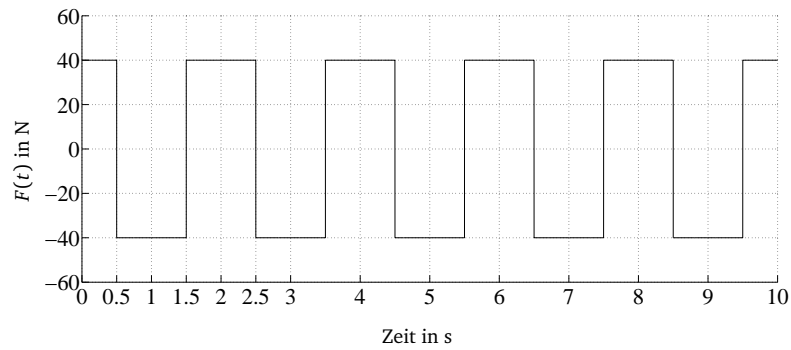


Abbildung 1.1: Rechteck-Signal $F(t)$.

Abbildung Lösung 5 zeigt die Simulation um den unteren Arbeitspunkt, Lösung 6 jene um den oberen.

Im unteren AP stimmen die beiden Modelle trotz der sehr großen Kraft und den großen Auslenkungen im Winkel von fast 2 rad gut überein. Bei der Simulation der Modelle im oberen AP fällt auf, dass das linearisierte Modell nur in einem sehr kleinen Bereich ausreichend genau ist. Beim nichtlinearen Modell fällt das Pendel herunter und schwingt um die untere Ruhelage.

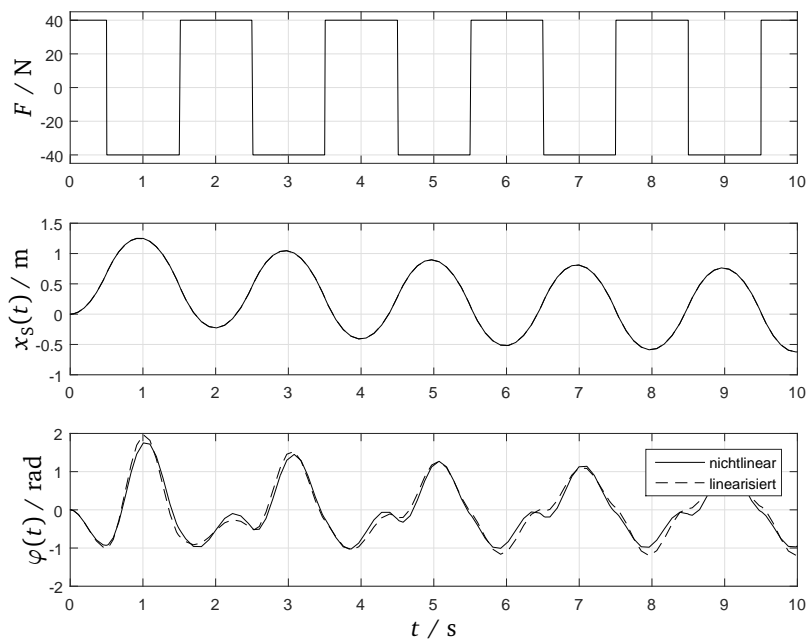


Abbildung Lösung 5: Vergleich des um $\Phi_0 = 0$ rad linearisierten und des nichtlinearen Modells bei Rechteckanregung

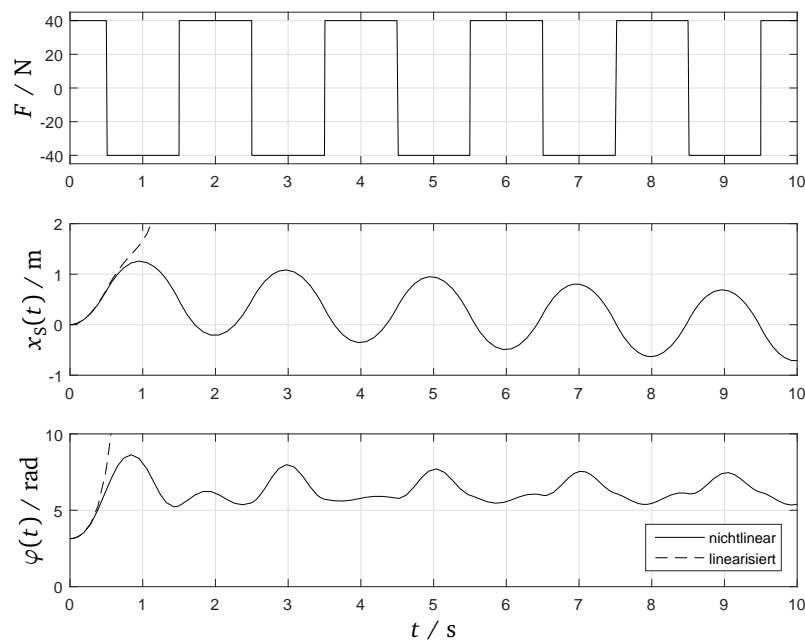


Abbildung Lösung 6: Vergleich des um $\Phi_0 = \pi \text{ rad}$ linearisierten und des nichtlinearen Modells bei Rechteckanregung

Aufgabe 1.7 (Durchführung/Nachbearbeitung):

Schließlich ist eine rechteckförmige Anregung gemäß Abbildung 1.2, welche bis auf eine Phasenverschiebung dem Signal in Abbildung 1.1 entspricht, anzunehmen.

- Welche Auswirkung hat diese Phasenverschiebung auf das Verhalten von $x_s(t)$ und $\varphi(t)$? Betrachten Sie auch hier beide Arbeitspunkte und erstellen zu jedem der beiden Arbeitspunkte Grafiken.

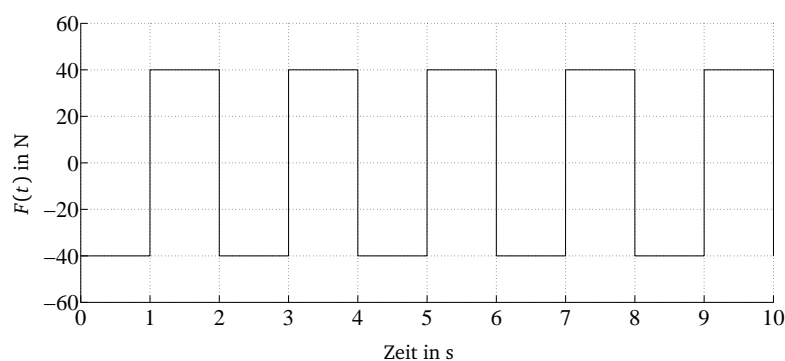


Abbildung 1.2: Rechteck-Signal $F(t)$.

Abbildung Lösung 7 zeigt die Simulation um den unteren Arbeitspunkt, Abbildung Lösung 8 jene um den oberen. Die Ergebnisse unterscheiden sich zunächst darin, dass der Schlitten durch die angreifende Kraft in die entgegengesetzte Richtung losfährt. Entsprechend beginnt das Pendel die Schwingung in die andere Richtung, bzw. fällt es in die andere Richtung herunter.

Zudem baut sich im zweiten Fall mit der verschobenen Anregung zunächst eine deutlich größere mittlere Geschwindigkeit des Schlittens auf, während im ersten Fall der Schlitten näherungsweise um $x_s = 0$ schwingt.

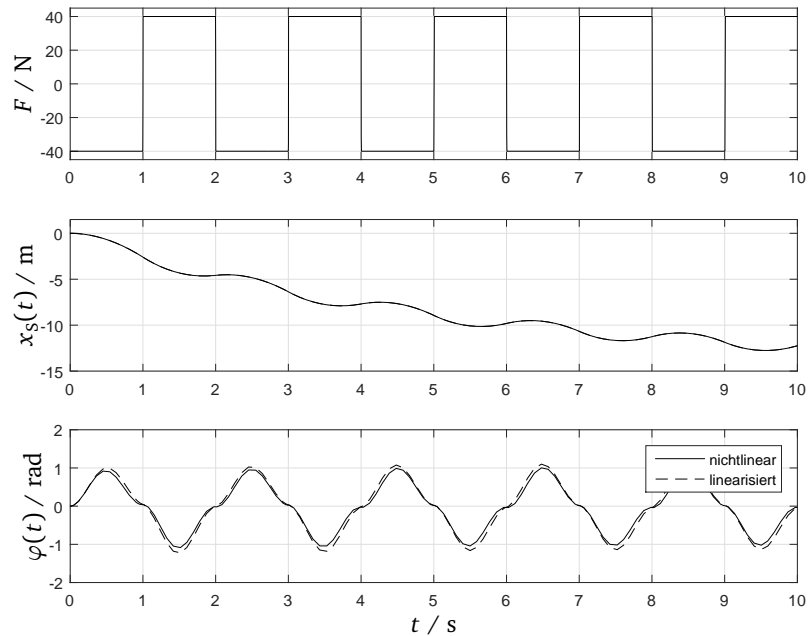


Abbildung Lösung 7: Vergleich des um $\Phi_0 = 0$ rad linearisierten und des nichtlinearen Modells bei phasenverschobener Rechteckanregung

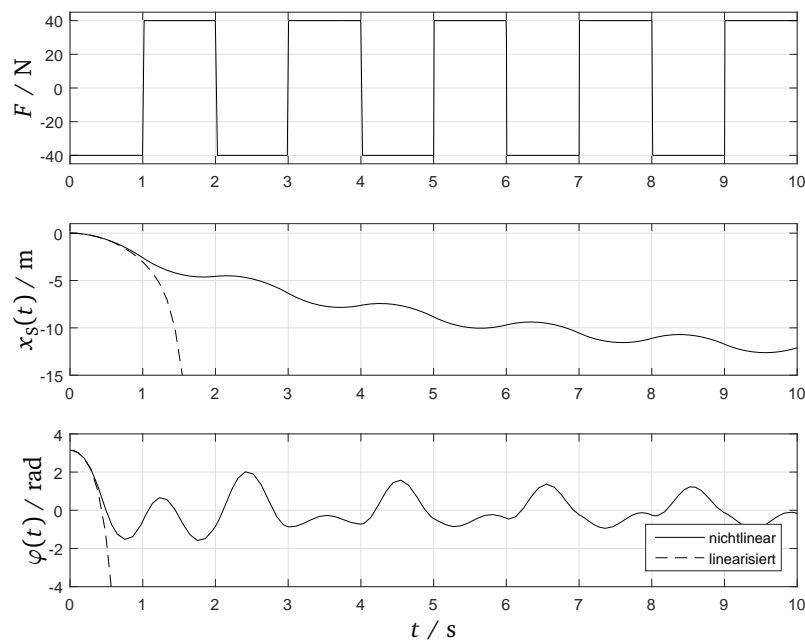


Abbildung Lösung 8: Vergleich des um $\Phi_0 = \pi \text{ rad}$ linearisierten und des nichtlinearen Modells bei phasenverschobener Rechteckanregung

1.2 WICHTIG: Hinweis zur Erstellung des Versuchsberichtes

Der Versuchsbericht (einer pro Gruppe) ist anhand der oben aufgeführten Fragen anzufertigen und innerhalb von einer Woche abzugeben.

Sowohl die Versuchsvorbereitung als auch die Versuchsdurchführung müssen vollständig im Bericht erscheinen. Dabei sollte ein besonderer Wert auf die Aussagekraft der Diagramme und der Screenshots gelegt werden. Die Simulationsergebnisse sollen als Diagramme mit der `plot`-Funktion (siehe Abschnitt Simulation, Erzeugen von Plots im Skript zu Versuch 1) dargestellt werden. Screenshots der *Scope*-Blöcke von Simulink sind für die schriftliche Dokumentation aus optischen Gründen ungeeignet.

Skalieren Sie die Diagramme sinnvoll, so dass das Ablesen bzw. Erkennen der interessierenden Größen möglich ist. In den Diagrammen ist eine eindeutige Achsenbezeichnung mit entsprechenden Größen und Einheiten unabdingbar. Bedenken Sie bei der Erstellung der Diagramme, dass diese auch schwarz/weiß ausgedruckt gut erkennbar und die einzelnen Graphen unterscheidbar sein müssen.

Strukturieren Sie die Protokolle so, dass auch nach einer längeren Zeitpause ein rascher Einstieg in die Thematik möglich ist. Dies kann im späteren Berufsleben eine erhebliche Zeitersparnis mit sich bringen.



Versuch 2

Steuerbarkeit und Beobachtbarkeit

2	Versuchsdurchführung	24
3	Progamme	31
3.1	Aufgabe 3 – Normalformen des Zustandsraummodells	31
3.2	Aufgabe 4 – Untersuchung der Steuer- und Beobachtbarkeit	32

2 Versuchsdurchführung

Die für die Durchführung der Versuche notwendige Zustandsraumdarstellung liegt bereits aus dem ersten Versuch vor (Modelle 1.0 und 1. π).

Zum Vergleich mit diesen Modellen soll die viskose Reibung des Pendelstabs vernachlässigt werden (Modelle 2.0 und 2. π). Des weiteren soll sämtliche Reibung vernachlässigt werden (Modelle 3.0 und 3. π).

Aufgabe 2.1 (Durchführung/Nachbearbeitung):

Vergleichen Sie die Zustandsraummodelle miteinander:

- Notieren Sie die Matrizen **A** und **B** aller Modelle.

$$\mathbf{A}_{1.0} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -0,1972 & 0,1834 & 0,0026 \\ 0 & 0 & 0 & 1 \\ 0 & 0,7215 & -36,5614 & -0,5238 \end{bmatrix}$$

$$\mathbf{B}_{1.0} = \begin{bmatrix} 0 \\ 0,1409 \\ 0 \\ -0,5153 \end{bmatrix}$$

$$\mathbf{A}_{1.\pi} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -0,1972 & 0,1834 & -0,0026 \\ 0 & 0 & 0 & 1 \\ 0 & -0,7215 & 36,5614 & -0,5238 \end{bmatrix}$$

$$\mathbf{B}_{1.\pi} = \begin{bmatrix} 0 \\ 0,1409 \\ 0 \\ 0,5153 \end{bmatrix}$$

$$\mathbf{A}_{2.0} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -0,1972 & 0,1834 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0,7215 & -36,5614 & 0 \end{bmatrix}$$

$$\mathbf{B}_{2.0} = \begin{bmatrix} 0 \\ 0,1409 \\ 0 \\ -0,5153 \end{bmatrix}$$

$$\mathbf{A}_{2.\pi} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -0,1972 & 0,1834 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -0,7215 & 36,5614 & 0 \end{bmatrix}$$

$$\mathbf{B}_{2.\pi} = \begin{bmatrix} 0 \\ 0,1409 \\ 0 \\ 0,5153 \end{bmatrix}$$

$$\mathbf{A}_{3,0} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0,1834 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -36,5614 & 0 \end{bmatrix}$$

$$\mathbf{B}_{3,0} = \begin{bmatrix} 0 \\ 0,1409 \\ 0 \\ -0,5153 \end{bmatrix}$$

$$\mathbf{A}_{3,\pi} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0,1834 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 36,5614 & 0 \end{bmatrix}$$

$$\mathbf{B}_{3,\pi} = \begin{bmatrix} 0 \\ 0,1409 \\ 0 \\ 0,5153 \end{bmatrix}$$

- Wodurch unterscheiden sich die Zustandsraumdarstellungen der Modelle?

Eingangsmatrizen

Die Eingangsmatrizen \mathbf{B} enthalten nicht die Parameter R_S und R_P , ihr Wert ist daher unabhängig von der Reibung.

Der Vergleich von oberem und unterem Arbeitspunkt zeigt, dass beide Vektoren betragsmäßig gleich sind, jedoch im vierten Eintrag ein unterschiedliches Vorzeichen aufweisen. Dies ist leicht zu erklären. Eine positive Kraft (also eine Kraft nach rechts) führt an beiden AP zu einer Beschleunigung des Schlittens nach rechts, daher ergibt sich hier das gleiche Vorzeichen. Die gleiche Kraft führt dazu, dass das Pendel sich nach links neigt, im unteren Arbeitspunkt im Uhrzeigersinn (daher das positive Vorzeichen), im oberen AP jedoch *gegen* den Uhrzeigersinn, was durch ein negatives Vorzeichen ausgedrückt wird.

Eine Kraft auf den Schlitten führt zu einer Beschleunigung des Schlittens \ddot{x}_S und Winkelbeschleunigung des Pendels $\ddot{\varphi}_S$. Es besteht jedoch kein Durchgriff auf die Geschwindigkeit (bzw. Winkelgeschwindigkeit), was physikalisch sinnvoll ist. Somit ergeben sich die Nulleinträge an erster und dritter Stelle.

Systemmatrizen

Die Systemmatrizen \mathbf{A} enthalten die Parameter R_S und R_P , ihr Wert ist daher abhängig von der Reibung.

Zuerst werden die Einträge $A_1(2,2)$ und $A_1(4,4)$ betrachtet. Sie wirken dämpfend auf das System, da mit

$$\ddot{x} = -0,1972 \cdot \dot{x} + \dots$$

$$\ddot{\varphi} = -0,5238 \cdot \dot{\varphi} + \dots$$

eine Geschwindigkeit (bzw. Winkelgeschwindigkeit) zu einer Beschleunigung (bzw. Winkelbeschleunigung) mit jeweils umgekehrten Vorzeichen führt.

Nun werden die Einträge $A_1(2,4)$ und $A_1(4,2)$ betrachtet. Über sie sind die beiden Bewegungen miteinander verkoppelt. Die Verkopplungen werden mit einem Gedankenexperiment erläutert:

Es wird beispielhaft angenommen, dass $u = 0$ und das Pendel senkrecht steht/hängt und der Schlitten mit Pendel sich nach rechts bewegt (Anfangsbedingung $x_0 = [0 \ 1 \ 0 \ 0]^T$):

Die Reibung zwischen Schlitten und Schlittenführung (R_s) führt – wie zuvor beschrieben – mit dem Eintrag $A_1(2, 2)$ zu einem Abbremsen des Schlittens. Diese Reibung führt über den Eintrag $A_1(4, 2)$ zu einer Winkelbeschleunigung des Pendels, das Pendel kippt nach rechts. (Für den oberen AP ergibt sich eine positive Winkelbeschleunigung, für den unteren eine negative.) Dieses Verhalten entspricht den Erwartungen.

Wie zuvor wird nun angenommen, dass $u = 0$ und der Schlitten in Ruhe ist, das Pendel senkrecht steht/hängt, aber sich im Uhrzeigersinn bewegt (Anfangsbedingung $x_0 = [0 \ 0 \ 0 \ 1]^T$):

Die Reibung im Pendellager (R_p) führt – wie zuvor beschrieben – mit dem Eintrag $A_1(4, 4)$ zu einem Abbremsen der Pendelbewegung. Diese Reibung führt über den Eintrag $A_1(2, 4)$ zu einer Beschleunigung des Wagens, er fährt im oberen AP nach rechts, im unteren nach links.

Werden die Reibungsfaktoren zu 0 gesetzt, so gehen die Dämpfungen der Bewegungen und die Verkopplungen der Bewegungen verloren.

Aufgabe 2.2 (Durchführung/Nachbearbeitung):

Vergleichen Sie die Eigenwerte der Zustandsraummodelle miteinander:

- Notieren sie die Eigenwerte aller Systeme.

$$\lambda_{1.0} = \begin{bmatrix} 0 \\ -0,1936 \\ -0,2637 + 6,0408j \\ -0,2637 - 6,0408j \end{bmatrix}$$

$$\lambda_{1.\pi} = \begin{bmatrix} 0 \\ -0,1936 \\ -6,3161 \\ 5,7887 \end{bmatrix}$$

$$\lambda_{2.0} = \begin{bmatrix} 0 \\ -0,1936 \\ -0,0018 + 6,0465j \\ -0,0018 - 6,0465j \end{bmatrix}$$

$$\lambda_{2.\pi} = \begin{bmatrix} 0 \\ -0,1936 \\ -6,0485 \\ 6,0448 \end{bmatrix}$$

$$\lambda_{3.0} = \begin{bmatrix} 0 \\ 0 \\ 6,0466j \\ -6,0466j \end{bmatrix}$$

$$\lambda_{3.\pi} = \begin{bmatrix} 0 \\ 0 \\ 6,0466 \\ -6,0466 \end{bmatrix}$$

- Welche Unterschiede liegen vor und worauf sind diese zurückzuführen? Gehen Sie dabei auf Realteil und Imaginärteil der Eigenwerte ein. Welche Aussagen über das Systemverhalten können Sie daraus ableiten?

Alle Systeme weisen (mindestens) einen Eigenwert in 0 auf. Dieser Integrator ist der Tatsache geschuldet, dass die Schlittenposition keine Rückführung aufweist. Es existiert z. B. keine Rückstellfeder.

Es werden nun die am unteren AP linearisierten Systeme betrachtet:

Das erste System (beide Reibungsfaktoren sind $\neq 0$) weist einen negativen reellen Eigenwert und ein konjugiert komplexes Eigenwertpaar mit negativem Realteil auf. Die horizontale Bewegung wird durch die Reibung im Schlitten gedämpft. Des Weiteren ist das Pendel schwingungsfähig, allerdings ist auch diese Bewegung gedämpft.

Wird die Reibung im Pendellager vernachlässigt, verändern sich die ersten beiden Eigenwerte nicht. Der Realteil des konjugiert komplexen Eigenwertpaars wird betragsmäßig klein. Die Reibung im Pendellager entfällt zwar, über die Verkopplung der Pendel- und Schlittenbewegungen führt jedoch die Reibung im Schlitten zu einer (recht geringen) Dämpfung von Pendelbewegungen.

Werden alle Reibungsfaktoren zu 0 gesetzt, ergibt sich ein Doppelintegrator und ein ungedämpftes Schwingen des Pendels.

Oberer AP:

Am oberen AP ist die Pendelbewegung instabil, es ergibt sich somit immer ein positiver Eigenwert. Alle Eigenwerte sind reell. Mit Vernachlässigung der Dämpfung verschiebt sich der instabile Eigenwert zunehmend nach rechts, was zu einer schnelleren Divergenz des Pendelwinkels führt.

Aufgabe 2.3 (Durchführung/Nachbearbeitung):

Normalformen des Zustandsraummodells:

- Schreiben Sie eine Funktion
 $[AD, BD, CD, DD] = \text{diagonalForm}(A, B, C, D)$
 die ein gegebenes System in Diagonalform transformiert (wenn möglich) und dokumentieren Sie diese in ihrem **Protokoll**.
- Transformieren Sie das System 1. π mit dieser Funktion auf Diagonalform.

$$A_D = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & -0,1936 & 0 & 0 \\ 0 & 0 & -6,3161 & 0 \\ 0 & 0 & 0 & 5,7887 \end{bmatrix}$$

$$B_D = \begin{bmatrix} 0,7143 \\ 0,7275 \\ 0,2809 \\ 0,2420 \end{bmatrix}$$

$$\mathbf{C}_D = \begin{bmatrix} 1 & -0,9818 & -0,0008 & 0,0008 \\ 0 & 0,0037 & -0,1564 & 0,1702 \end{bmatrix}$$

$$\mathbf{D}_D = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

- Transformieren Sie das System 1. π mit der Funktion canon auf Diagonalform.

$$\mathbf{A}_D = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & -0,1936 & 0 & 0 \\ 0 & 0 & -6,3161 & 0 \\ 0 & 0 & 0 & 5,7887 \end{bmatrix}$$

$$\mathbf{B}_D = \begin{bmatrix} 0,3571 \\ 0,6584 \\ 0,3284 \\ 0,2899 \end{bmatrix}$$

$$\mathbf{C}_D = \begin{bmatrix} 2 & -1,0848 & -0,0007 & 0,0007 \\ 0 & 0,0041 & -0,1337 & 0,1421 \end{bmatrix}$$

$$\mathbf{D}_D = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

- Welche Unterschiede sind zu erkennen? Wie lassen Sie sich erklären?

Beide Funktionen transformieren das System aus Diagonalform. Aufgrund numerischer Ungenauigkeiten bei der ersten Berechnung stehen in der Matrix \mathbf{A}_D auch außerhalb der Hauptdiagonalen Einträge, die von 0 verschieden sind. Sie sind jedoch betragsmäßig klein, dass sie vernachlässigt werden können.

Die Eingangs- und Ausgangsmatrizen der beiden berechneten Systeme unterscheiden sich stark. Dies lässt sich dadurch erklären, dass die Funktion canon das System vor der Durchführung der Modaltransformation skaliert, um die numerischen Eigenschaften zu verbessern. Die Diagonalform ist nicht eindeutig. Aufgrund der Anschauung lässt sich jedoch leicht eine Transformation finden, mit der sich beide Systeme vergleichen lassen. Werden beide Systeme mit der Transformationsmatrix $\mathbf{T} = \text{diag}(\mathbf{B}_D)$ transformiert, so ergeben sich jeweils Eingangsmatrizen der Form $\tilde{\mathbf{B}}_D = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}^T$ und die Ausgangsmatrizen sind identisch.

Da das System nur reelle Eigenwerte besitzt, stehen auf der Hauptdiagonalen der Systemmatrix nur reellwertige Einträge.

- Transformieren Sie das System 1.0 auf Modalform.

$$\mathbf{A}_D = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & -0,1936 & 0 & 0 \\ 0 & 0 & -0,2637 & 6,0408 \\ 0 & 0 & -6,0408 & -0,2637 \end{bmatrix}$$

$$\mathbf{B}_D = \begin{bmatrix} 0,3571 \\ -0,6585 \\ -0,5137 \\ 0,0562 \end{bmatrix}$$

$$\mathbf{C}_D = \begin{bmatrix} 2 & 1,0847 & 0 & 0,0008 \\ 0 & -0,0042 & -0,0127 & -0,1647 \end{bmatrix} \quad \mathbf{D}_D = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Die nachfolgenden Aufgaben sind in Ihrem Protokoll zu beantworten.

Aufgabe 2.4 (Durchführung/Nachbearbeitung):

Die Zustandsraumbeschreibung des Schlitten-Pendel-Systems 1. π am oberen Arbeitspunkt soll nun anhand der beschriebenen Kriterien auf die vollständige Steuer- und Beobachtbarkeit untersucht werden. Hierzu sollen die angegebenen formellen Zusammenhänge verwendet werden. Schreiben Sie die Funktionen, so dass sie *möglichst allgemein* verwendbar sind, also z. B. Systeme beliebiger Ordnung verarbeiten können. Ebenfalls sollen von MATLAB zur Verfügung gestellte Funktionen angewendet werden.

- Prüfen Sie die Steuer- und Beobachtbarkeit des Systems 1. π nach Kalman. Schreiben Sie hierzu die Funktionen
 - `checkCtrbKalman(A, B)` und
 - `checkObsvKalman(A, C)`

und verifizieren Sie die Ergebnisse mit den von MATLAB zur Verfügung gestellten Funktionen `obsv` und `ctrb`. Dokumentieren Sie sowohl die beiden Funktionen als auch die Ergebnisse des Vergleichs in Ihrem Protokoll.

- Prüfen Sie die Steuer- und Beobachtbarkeit des Systems 1. π nach Gilbert. Schreiben Sie hierzu die Funktionen
 - `checkCtrbGilbert(A, B)` und
 - `checkObsvGilbert(A, C)`.

Dokumentieren Sie die beiden Funktionen in Ihrem Protokoll.

- Prüfen Sie die Steuer- und Beobachtbarkeit des Systems 1. π nach Hautus. Schreiben Sie hierzu die Funktionen
 - `checkCtrbHautus(A, B)` und
 - `checkObsvHautus(A, C)`.

Dokumentieren Sie die beiden Funktionen in Ihrem Protokoll.

- Welche Aussagen lassen sich über das System machen?
- Können die Kriterien auch auf die Systeme 2. π und 3. π angewendet werden?



3 Programme

3.1 Aufgabe 3 – Normalformen des Zustandsraummodells

Der MATLAB-Code zur Transformation des Systems ist im Listing 3.1 dargestellt.

Listing 3.1: Funktion der Aufgabe 3

```
function Aufgabe_Normalformen
    clc;

4     para

        %% oberer AP
        disp( 'oberer_AP' );
        [A,B,C,D] = initLinear( 1, mp, ms, Rs, Rp, g, pi );
9     sys = ss( A, B, C, D );

        % Diagonalform
        disp( 'diagonalForm' );
        [AD, BD, CD, DD] = diagonalForm( A, B, C, D )
14

        disp( 'canon' );
        sysD = canon( sys, 'modal' );
        [AD, BD, CD, DD] = ssdata( sysD )

19     %% unterer AP
        disp( 'unterer_AP' );
        [A,B,C,D] = initLinear( 1, mp, ms, Rs, Rp, g, 0 );
        sys = ss( A, B, C, D );

24     % Modalform
        disp( 'canon' );
        sysD = canon( sys, 'modal' );
        [AD, BD, CD, DD] = ssdata( sysD )

29 end
```

Der MATLAB-Code der Funktion diagonalForm ist im Listing 3.2 dargestellt.

Listing 3.2: Funktion der Aufgabe 3

```
1 function [AD, BD, CD, DD] = diagonalForm( A, B, C, D )

    n = size( A, 1 );
```

```

6      % Eigenwerte und Eigenvektoren berechnen
      [V, lambda] = eig( A );

      % prüfen, ob Eigenvektoren normiert sind
      for i = 1:n
          V(:,i).' * V(:,i)
11      end

      if ( rank( V ) ~= n )
          error( 'System_list_nicht_diagonalähnlich' );
      end

16      AD = V \ A * V; % =inv( V ) * A * V;
      BD = V \ B; % = inv( V ) * B;
      CD = C * V;
      DD = D;

21      end

```

Zur Transformation des Zustandsraummodells in die Diagonalform werden die Eigenwerte λ_i und die Eigenvektoren v_i der Systemmatrix A benötigt. Die Transformationsmatrix T beinhaltet demnach alle normierten Eigenvektoren.

3.2 Aufgabe 4 – Untersuchung der Steuer- und Beobachtbarkeit

Der MATLAB-Code zur Überprüfung des Steuer- und Beobachtbarkeit des Systems ist im Listing 3.3 dargestellt.

Listing 3.3: Funktion der Aufgabe 4

```

function Aufgabe_SteuerbarkBeobachtbark
    % Untersuchung der Steuer- und Beobachtbarkeit nach verschiedenen
    % Kriterien
3      clc;

    para

8      % [A,B,C] = initLinear( l, mp, ms, Rs, Rp, g, pi );
      % [A,B,C] = initLinear( l, mp, ms, Rs, 0, g, pi );
      [A,B,C] = initLinear( l, mp, ms, 0, 0, g, pi );

      %% Steuer- und Beobachtbarkeit nach Kalman
13      checkCtrbKalman( A, B );
      % if ( rank( ctrb( A, B ) ) == size( A, 1 ) )
      %     disp( 'Steuerbar nach Kalman!' );
      % else
      %     disp( 'Nicht steuerbar nach Kalman!' );
18      % end

      checkObsvKalman( A, C );

```

```

%      if ( rank( obsv( A, C ) ) == size( A, 1 ) )
%          disp( 'Beobachtbar nach Kalman!' );
23 %      else
%          disp( 'Nicht beobachtbar nach Kalman!' );
%      end

      %% Steuer- und Beobachtbarkeit nach Gilbert
28 checkCtrbGilbert( A, B );
   checkObsvGilbert( A, C );

      %% Steuer- und Beobachtbarkeit nach Hautus
33 checkCtrbHautus( A, B );
   checkObsvHautus( A, C );

end

```

Die Ausgabe ist im Listing 3.4 dargestellt.

Listing 3.4: Ausgabe Aufgabe 4

```

Steuerbar nach Kalman!
Steuerbar nach Kalman!
Beobachtbar nach Kalman!
Beobachtbar nach Kalman!
5 Prüfen, ob es eine Nullzeile gibt:

BT =

    196.0784
10    196.0781
    0.7234
    0.7052

Steuerbar nach Gilbert!
15 Prüfen, ob es eine Nullspalte gibt:

CT =

    1.0000    -1.0000   -0.3504    0.3576
20         0     0.0000   -0.1118    0.1143

Beobachtbar nach Gilbert!
Eigenwert 0 -> Rang = 4
Eigenwert -0.003234 -> Rang = 4
25 Eigenwert -2.5281 -> Rang = 4
Eigenwert 2.4688 -> Rang = 4
Steuerbar nach Hautus!
Eigenwert 0 -> Rang = 4
Eigenwert -0.003234 -> Rang = 4
30 Eigenwert -2.5281 -> Rang = 4
Eigenwert 2.4688 -> Rang = 4

```

Beobachtbar nach Hautus!

3.2.1 Steuer- und Beobachtbarkeit nach KALMAN

In den Listings 3.5 und 3.6 werden die Kriterien von Kalman überprüft.

Listing 3.5: Steuerbarkeit nach Kalman

```
function checkCtrbKalman(A, B)
    n = size( A, 1 );
3    MS = B;
    for i = 2:n
        MS = [B A*MS];
    end

8    if ( rank( MS ) == n )
        disp( 'Steuerbar_nach_Kalman!' );
    else
        disp( 'NICHT_steuerbar_nach_Kalman!' );
    end
13 end
```

Listing 3.6: Beobachtbarkeit nach Kalman

```
function checkObsvKalman(A, C)
2    n = size( A, 1 );
    MB = C;
    for i = 2:n
        MB = [C; MB*A];
    end

7    if ( rank( MB ) == n )
        disp( 'Beobachtbar_nach_Kalman!' );
    else
        disp( 'NICHT_beobachtbar_nach_Kalman!' );
12    end
end
```

Die Steuerbarkeits- bzw. Beobachtbarkeitsmatrix wird in einer Schleife aufgebaut, so dass beliebige Systemordnungen überprüft werden können.

Das Ergebnis dieser beiden Funktionen wurde mit Listing 3.7 überprüft.

Listing 3.7: Prüfung der Steuer- und Beobachtbarkeit nach Kalman

```
if ( rank( ctrb( A, B ) ) == size( A, 1 ) )
2    disp( 'Steuerbar_nach_Kalman!' );
else
    disp( 'Nicht_steuerbar_nach_Kalman!' );
end
```

```

7 if ( rank( obsv( A, C ) ) == size( A, 1 ) )
    disp( 'Beobachtbar_nach_Kalman!' );
else
    disp( 'Nicht_beobachtbar_nach_Kalman!' );
end

```

3.2.2 Steuer- und Beobachtbarkeit nach GILBERT

In den Listings 3.8 und 3.9 werden die Kriterien von Gilbert überprüft.

Listing 3.8: Steuerbarkeit nach Gilbert

```

function checkCtrbGilbert(A, B)
    n = size( A, 1 );

4
    [v, lambda] = eig( A );
    T = v;

    % Prüfen, ob es nur einfache Eigenwerte gibt
9    if ( length( unique( diag( lambda ) ) ) ~= n )
        disp( 'Es_existieren_mehrfache_Eigenwerte!' );
        return;
    end

14    disp( 'Prüfen, ob es eine Nullzeile gibt:' );
    BT = T \ B % = inv( T ) * B
    if ( all( any( BT, 2 ) ) )
        disp( 'Steuerbar_nach_Gilbert!' );
    else
19    disp( 'NICHT_steuerbar_nach_Gilbert!' );
    end
end

```

Listing 3.9: Beobachtbarkeit nach Gilbert

```

function checkObsvGilbert(A, C)
    n = size( A, 1 );

4
    [v, lambda] = eig( A );
    T = v;

    % Prüfen, ob es nur einfache Eigenwerte gibt
9    if ( length( unique( diag( lambda ) ) ) ~= n )
        disp( 'Es_existieren_mehrfache_Eigenwerte!' );
        return;
    end

```

```

14     disp( 'Prüfen, ob es eine Nullspalte gibt:' )
        CT = C * T
        if ( all( any( CT, 1 ) ) )
            disp( 'Beobachtbar nach Gilbert!' );
        else
19         disp( 'NICHT beobachtbar nach Gilbert!' );
        end
    end
end

```

Zuerst werden die Eigenwerte berechnet und geprüft, ob es sich um paarweise verschiedene Eigenwerte handelt. Wenn ja, kann das System auf Diagonalform gebracht werden und durch Prüfung von `all(any(CT, 1))` oder `all(any(BT, 2))` geprüft werden, ob keine Nullzeilen bzw. Nullspalten vorliegen.

Der Fall mehrfacher Eigenwerte wird nicht betrachtet.

3.2.3 Steuer- und Beobachtbarkeit nach Hautus

In den Listings 3.10 und 3.11 werden die Kriterien von Hautus überprüft.

Listing 3.10: Steuerbarkeit nach Hautus

```

function checkCtrbHautus(A, B)
    n = size( A, 1 );

4     % nach Hautus
        ew = eig( A );
        OK = true;
        for i = 1:n
            rg = rank( [ eye(n)*ew(i) - A, B ] );
9            disp( ['Eigenwert_', num2str( ew(i) ), ' -> Rang_', num2str( rg -
                <-) ] );
            if ( rg ~= n )
                OK = false;
            end
        end
    end

14     if ( OK )
        disp( 'Steuerbar nach Hautus!' );
    else
        disp( 'NICHT steuerbar nach Hautus!' );
    end
19     end
end

```

Listing 3.11: Beobachtbarkeit nach Hautus

```

function checkObsvHautus(A, C)
    n = size( A, 1 );

```

```

% nach Hautus
5  ew = eig( A );
    OK = true;
    for i = 1:n
        rg = rank( [ eye(n)*ew(i) - A; C ] );
        disp( ['Eigenwert_', num2str( ew(i) ), '_->_Rang_=_', num2str( rg ->
10         <= n ) ] );
        if ( rg ~= n )
            OK = false;
        end
    end

15  if ( OK )
        disp( 'Beobachtbar_nach_Hautus!' );
    else
        disp( 'NICHT_beobachtbar_nach_Hautus!' );
    end

20 end

```

Die Überprüfung des Kriterium erfolgt für alle Eigenwerte des Systems in einer for-Schleife.

3.2.4 Aussagen über das System

Das System ist also vollständig steuer- und beobachtbar.

3.2.5 Anwendung der Kriterien auf andere Systeme

Die Kriterien können problemlos auch auf das System 2.π angewendet werden.

Das System 3.π besitzt einen doppelten Eigenwert in 0. Die Eigenvektoren sind nicht mehr linear unabhängig und somit ist **T** nicht regulär. Das System kann daher nicht auf Diagonalform transformiert werden, das Kriterium nach Gilbert ist nicht anwendbar. Mit Hilfe der anderen Kriterien kann jedoch die vollständige Steuer- und Beobachtbarkeit nachgewiesen werden.



Versuch 3

LQ-Regelung und Animation

4	Versuchsdurchführung	40
5	Programme	47
5.1	Aufgabe 5	47
5.2	Aufgabe 6 und Zusatzaufgabe	48
6	Protokoll	52
7	s-Function systemPendel_V3	61

4 Versuchsdurchführung

Ziel dieses Versuches ist es, einen LQ-Reglerentwurf für das Schlitten-Pendel-Modell durchzuführen und die Wirkung der Gewichtungsmatrizen \mathbf{Q} und \mathbf{R} zu untersuchen. Die Simulationsergebnisse sollen in einer Animation dargestellt werden. Die folgenden Aufgaben behandeln dabei jeweils einen Einzelschritt zum Erreichen dieses Zieles.

Grundsätzlich können Sie auch eigene Wege gehen. Jedoch werden die verwendeten Funktionen auch teilweise im nächsten Versuch benötigt, so dass Sie sich unnötige Arbeit sparen, wenn die Funktionen die hier angegebene Syntax besitzen.

Im folgenden wird die viskose Reibung in der Führung des Wagens und im Pendelgelenk vernachlässigt ($R_s = R_p \equiv 0$).

Aufgabe 3.1 (Durchführung/Nachbearbeitung):

- Vervollständigen Sie die Funktion

```
function stPendel = ladePendel(),
```

welche eine Struktur, die die Parameter des Schlitten-Pendel-Systems enthält, zurückgibt. Für eine reibungslose Zusammenarbeit mit gegebenen Funktionen in den folgenden Versuchen sollte diese Struktur die Felder

```
stPendel =  
    lPendel: 0.4100  
    mPendel: 0.1770  
    mSchlitten: 7.0550  
    g: 9.8100
```

besitzen. (Wenn Sie im Laufe der weiteren Programmierung zusätzliche Felder hinzufügen wollen, dann können Sie das gerne machen.)

```
function stPendel = ladePendel()  
  
    stPendel.lPendel = .41;  
    stPendel.mPendel = .177;  
    stPendel.mSchlitten = 7.055;  
    stPendel.g = 9.81;  
  
end % function ladePendel
```

Aufgabe 3.2 (Durchführung/Nachbearbeitung):

- Vervollständigen Sie die Funktion

`function [A, B, C, D] = linPendelZR(stPendel, AP),`

welche in Abhängigkeit der Pendeldata und des Arbeitspunkts (untere oder obere Ruhelage) die in Versuch 1 linearisierten Matrizen **A**, **B**, **C** und **D** zurückgibt. (AP kann dabei 0 oder π sein.)

Diese Funktion dient dazu, später schnell den Arbeitspunkt wechseln zu können, ohne anzufangen im Code Teile aus- und einzukommentieren.

Wichtig: Der Zustandsvektor soll folgendermaßen aufgebaut sein:

$$\mathbf{x} = \begin{bmatrix} \text{Weg Schlitten} \\ \text{Geschwindigkeit Schlitten} \\ \text{Winkel Pendel} \\ \text{Winkelgeschwindigkeit Pendel} \end{bmatrix}.$$

```
function [A, B, C, D] = linPendelZR(stPendel, AP)

% Fehlerabfrage
if (AP~=0 && AP~=pi)
    fprintf('-----\n');
    fprintf('Es wurde ein falscher Arbeitspunkt gewählt!!\n');
    fprintf('Möglicher Arbeitspunkt 0 bzw pi\n');
    fprintf('-----\n');
    A='Error';
    B='Error';
    C='Error';
    D='Error';

    return;
end;

% Bestimmung der ZR-Darstellung
% Abkürzungen
mP = stPendel.mPendel;
mS = stPendel.mSchlitten;
lP = stPendel.lPendel;
g = stPendel.g;

% Abfrage um welchen Punkt linearisiert wird.
switch (AP)
    case 0

        A = [
            [0 1 0 0];
            [0 0 3*g*mP 0] / (4*mS+mP);
            [0 0 0 1];
            [0 0 -6*g*(mS+mP) 0] / (lP*(4*mS+mP));
        ];

        B = [0; 4; 0; -6/lP]/(4*mS+mP);

        C = [1 0 0 0; 0 0 1 0];

        D = [0; 0];
```

```

case pi
    A = [
        [0 1 0 0];
        [0 0 3*g*mP 0] / (4*mS+mP);
        [0 0 0 1];
        [0 0 6*g*(mS+mP) 0] / (1P*(4*mS+mP));
    ];

    B = [0; 4; 0; 6/1P]/(4*mS+mP);

    C = [1 0 0 0; 0 0 1 0];

    D = [0; 0];

end % switch (AP)

end % function linPendelZR

```

Aufgabe 3.3 (Durchführung/Nachbearbeitung):

- Vervollständigen Sie die Funktion

$[K, \text{PoleRK}] = \text{berechneLQR}(A, B, Q, R),$

die die Reglermatrix K mittels des LQR-Verfahren berechnet und diese zurückgibt. In Hinblick auf Versuch 4 soll die Funktion ebenfalls die Pole des *geschlossenen* Regelkreises zurückgeben.

```

function [K, poleRK] = berechneLQR(A, B, Q, R)

% Fehlerabfragen:
K = 'Error';
poleRK = [];

% Steuerbarkeit
if ( rank(ctrb(A, B)) ~= length(A) )
    fprintf('Das System ist nicht steuerbar - LQR-Verfahren nicht anwendbar -> \n');
    return;
end

% Test auf Symmetrie von Q:
if ( any(any(Q ~= Q.')) )
    fprintf('Q muss symmetrisch gewählt werden!');
    return;
end

% Test auf positive Definitheit von Q:
if ( any( eig(Q) <= 0 ) )
    fprintf('Q muss positiv definit gewählt werden!');
    return;
end

```

```

% Test auf positive Definitheit von R:
if ( any( eig(R) <= 0 ) )
    fprintf('R_muss_positiv_definit_gewählt_werden!');
    return;
end

% Reglerberechnung:
[K, ~, poleRK] = lqr(A, B, Q, R);
% poleRK: Pole geschlossener Regelkreis

end % function berechneLQR

```

Der **K**-Vektor wird durch die MATLAB-Funktion `lqr` bestimmt. Die Pole des geschlossenen Regelkreises werden ebenfalls von dieser Funktion berechnet. Alternativ könnten die Pole nach der Berechnung von **K** über `poleRK = eig(A-B*K);` bestimmt werden.

Aufgabe 3.4 (Durchführung/Nachbearbeitung):

- Implementieren Sie die Regelung in Simulink und vervollständigen Sie das unten stehende Strukturbild. Für das Schlitten-Pendel-System soll dabei das nichtlineare Modell verwendet werden. Verwenden Sie dazu die gegebene s-Function „systemPendel_V3“. Diese unterscheidet sich in folgenden Punkten von der s-Function, die Sie in Versuch 1 erstellt haben:
 - Es werden alle vier Zustände zurückgegeben. (In der in Aufgabe 2 angegebenen Reihenfolge!)
 - In den Parametern wird für die Pendeldaten die oben beschriebene Struktur benutzt.
 - Es wird keine viskose Reibung berücksichtigt.
 - Es wird in den Parametern auch der Anfangswert angegeben.

Natürlich können Sie diese Änderungen auch selber vornehmen. In Abbildung 4.1 ist die Verwendung gezeigt.

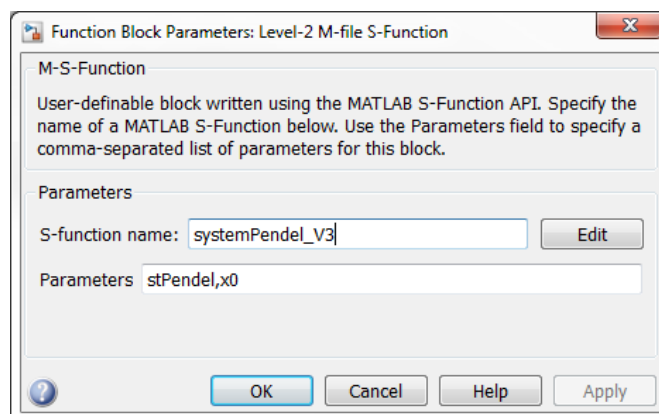


Abbildung 4.1: Verwendung der s-Function „systemPendel_V3“

Das Modell sollte die Reglermatrix **K**, die Pendeldaten, den Arbeitspunkt und den Anfangszustand \mathbf{x}_0 aus dem Workspace lesen und den Verlauf der Zustände $\mathbf{x}(t)$ mit dem Zeitvektor in den Workspace schreiben.

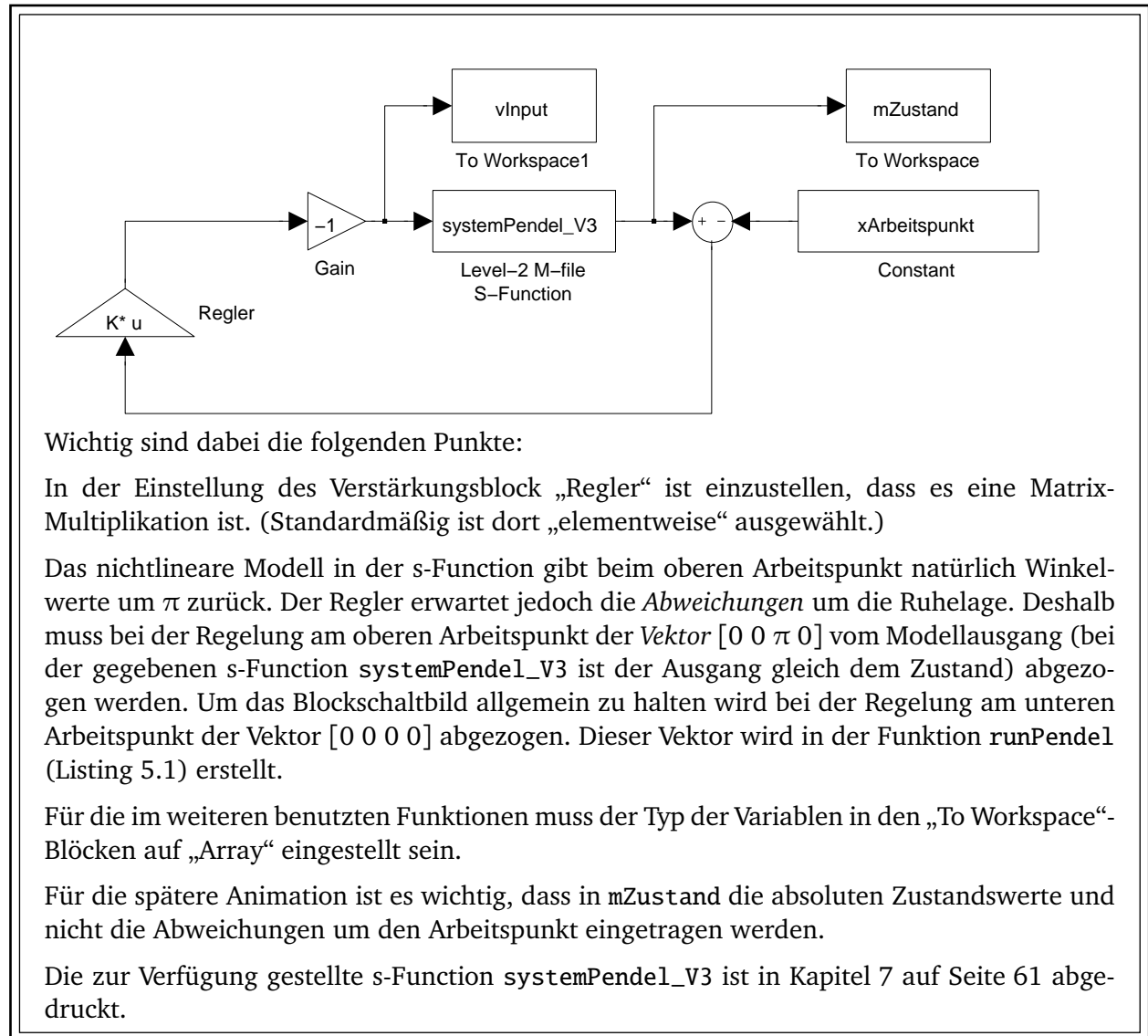
Hinweise:

Überlegen Sie, welche Werte das Modell des Systems für die einzelnen Arbeitspunkte zurückgibt,

und welche der Regler erwartet.

Um einen Sollgrößenprung auf das System zu geben, benutzen Sie die Anfangswerte. Geregelt wird das System immer in die Lage $x = 0$.

Testen Sie ihr Modell dadurch, dass Sie einen Anfangswert ungleich 0 vorgeben und interpretieren Sie den Ausgang.



Die nachfolgenden Aufgaben sind in Ihrem Protokoll zu beantworten.

Aufgabe 3.5 (Durchführung/Nachbearbeitung):

- Schreiben Sie eine Funktion

`[vT, mX] = runPendel(stPendel, AP, K, x0)`

die die Pendelraten, die Reglermatrix und die Anfangswerte des Systems übergeben bekommt, und das Simulinkmodell ausführt. Diese Funktion soll den Verlauf der Zustandsgrößen mit dem dazugehörigen Zeitvektor zurückgeben.

Fügen Sie bitte (unkommentiert) den Code Ihrer Funktion `runPendel` Ihrem Protokoll bei.

Aufgabe 3.6 (Durchführung/Nachbearbeitung):

- Schreiben Sie eine Funktion

`animierePendel(vT, mX, stPendel, hAxes)`

zur Animation des Pendelschlittens. Diese Funktion soll den Verlauf der Zustandsgrößen sowie den dazugehörigen Zeitvektor und die Pendelraten übernehmen. Zudem soll diese Funktion in Hinblick auf Übung 4 auch das Handle des Achsensystems übernehmen, in das die Animation gezeichnet werden soll. Wenn diese Variable leer ist, dann soll ein neues Figure mit Achsensystem erstellt werden.

Testen Sie Ihre Funktion mit Simulationsergebnissen.

Hinweise:

Mit `isempty(Variablenname)` kann überprüft werden, ob eine Variable eine leere Matrix enthält.

Orientieren Sie sich beim Programmieren an dem Beispiel `animierePunkt` von Listing 11.3.

Es wäre praktisch, im Titel des Achsensystems die aktuelle Simulationszeit anzuzeigen.

Betrachten Sie sich den Zeitvektor `vT`. Wurde ein Löser mit variabler Schrittweite verwendet, so sind die Simulationszeitpunkte nicht äquidistant, die Pause zwischen zwei Bildern der Animation müsste für jedes Bild neu bestimmt werden. Es ist sinnvoll, die Daten für die Animation in einer vernünftigen konstanten Bildrate vorliegen zu haben. Hierzu kann die Funktion `xi = interp1(x,y,xi)` verwendet werden. Ein Vektor mit äquidistanten Zeitpunkten kann leicht mit `vTAnim = 0:pause:vT(end)` erzeugt werden.

In Abbildung 4.2 ist beispielhaft gezeigt, wie so eine Animation aussehen könnte. Es muss allerdings nicht so aufwendig (mit Kästen für Schlitten) ausgeführt werden, es reicht auch eine einzige bewegte Linie als Pendelstab aus.

Aufgabe 3.7 (Durchführung/Nachbearbeitung):

- Verändern Sie die Einträge in den Matrizen **Q** und **R** und interpretieren Sie das sich daraus ändernde Systemverhalten. Achten Sie hierbei darauf, dass die Anforderungen an die Matrizen **Q** und **R** (Symmetrie, positive Definitheit) stets erfüllt sind.

Hinweise:

Um die Einflüsse der einzelnen Parameter auf den Reglerentwurf beurteilen zu können, sollten Sie nur jeweils einen Parameter verändern.

Für die direkte Interpretation der Ergebnisse stellt die Animation eine Hilfe dar. Für das Protokoll sind jedoch Plots der Zustände über der Zeit sinnvoller.

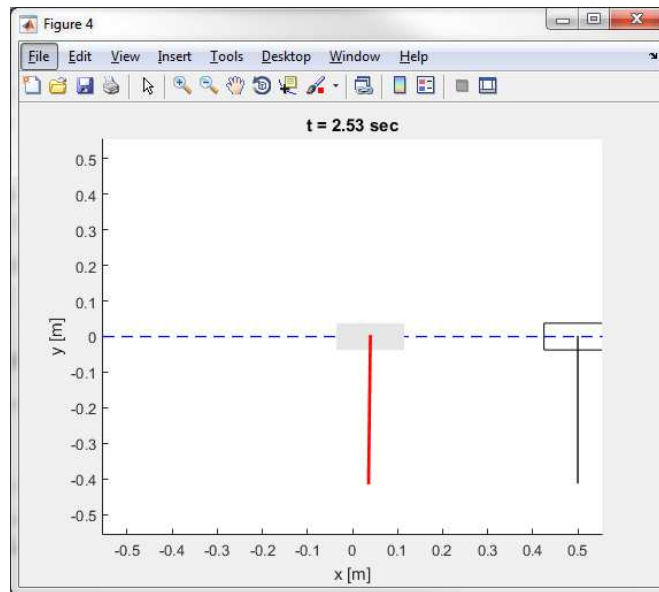


Abbildung 4.2: Beispiel für Animation

Simulieren Sie das geregelte System für die untere und obere Ruhelage und wählen Sie jeweils mindestens zwei aussagekräftige Simulationsergebnisse bei verschiedenen Gewichtungsmatrizen aus. Stellen Sie die Simulationsergebnisse von $F(t)$, $x_s(t)$ und $\varphi(t)$ dar und kommentieren Sie diese.

Aufgabe 3.8 (Durchführung/Nachbearbeitung):

- Erweitern Sie Ihre Funktion `animierePendel`, so dass, wenn gewünscht, ein avi-Video generiert wird. Dazu können Sie ein weiteres Argument hinzufügen. Bedenken Sie dabei, dass die Funktion dennoch mit dem oben angegebenen Aufruf funktionieren sollte, da dies im nächsten Versuch benötigt wird.

Hinweis:

Mit dem Befehl `nargin` können Sie in einer Funktion die Anzahl der tatsächlich übergebenen Parameter bestimmen.

Fügen Sie bitte (unkommentiert) den Code Ihrer Funktion `animierePendel` und einen Screenshot der Animation Ihrem Protokoll bei.

5 Programme

5.1 Aufgabe 5

Zusätzlich zu den in der Aufgabenstellung angegebenen Rückgabewerten gibt die hier verwendete Funktion noch den Verlauf der Eingangsgröße im Vektor `vU` zurück.

Listing 5.1: `runPendel`

```
function [vT, mX, vU] = runPendel(stPendel, AP, K, x0)

    % "arbeitspunkt" ist Winkel in rad
    xArbeitspunkt = [0, 0, AP, 0];

5    % Für Modell nötige Variablen im Base-Workspace anlegen
    assignin('base', 'xArbeitspunkt', xArbeitspunkt);

    assignin('base', 'stPendel', stPendel);
10    assignin('base', 'x0', x0);

    assignin('base', 'K', K);

    % Simulation ausführen (hier mit fester Dauer von 10 Sekunden)
15    % Modellierte Zeitpunkte in "vT" speichern
    vT = sim('Regelung', 10);

    % Modell schreibt "mZustand" in den Workspace.
    % Dieses als "mX" zurückgeben.
20    mX = mZustand;
    % Entsprechend "vInput" als "vU" zurückgeben.
    vU = vInput;

end % function runPendel
```

Das folgende Skript verwendet alle bisher erstellten Funktionen sowie die Animations-Funktion der nächsten Aufgabe.

Listing 5.2: Versuch3

```
% Skript zum Ausführen der in Versuch 3 erstellten Funktionen
clear all
clc
stPendel = ladePendel();

5    % Hängendes Pendel AP = 0
    % Stehendes Pendel AP = pi
```

```

AP = 0;
10 % Anfangswerte
x0 = [0.5 0 1*AP 0];

15 % Gewichtung Q, R

Q = diag([100 1 100 1]);

R = 1;
20

[A, B, C, D] = linPendelZR(stPendel, AP);

[K, poleRK] = berechneLQR(A, B, Q, R);
25

[vT, mX, vU] = runPendel(stPendel, AP, K, x0);

% Animation ohne Aufnahme eines Videos
animierePendel(vT, mX, stPendel, [], 0.025);
30 % % Animation mit Aufnahme eines Videos
% vFrames = animierePendel(vT, mX, stPendel, [], 0.025, 1);
% % Abspielen Video
% movie(ffigure(), vFrames, 1, 10)

35 % Plotten der Verläufe u(t), x(t), phi(t)
plotResults(vT, mX, vU, Q, R);

```

5.2 Aufgabe 6 und Zusatzaufgabe

Die letzten beiden Argumente von dieser Version von `animierePendel` sind optional, so dass diese Funktion genau wie in der Aufgabenstellung beschrieben werden kann. Wenn das letzte Argument `rec` den Wert 1 besitzt, dann wird ein Vektor mit Frame-Strukturen zurückgegeben. (Dabei wird die Anzahl der Frames auf 500 begrenzt. Liegen mehr als 500 Zeitpunkte vor, dann wird nicht jeder Frame in die Struktur gespeichert.)

Es sollte darauf geachtet werden, dass die Achsenskalierung in beide Richtungen gleich ist, damit das Pendel während der Bewegung nicht verzerrt dargestellt wird.

Listing 5.3: `animierePendel`

```

function vFrames = animierePendel(vT, mX, stPendel, hAxes, p, rec)
% t      := Zeitvektor
% x      := Zustandsvektor
4 %      x1 = Weg des Schlittens
%      x2 = Beschleunigung des Schlittens
%      x3 = Winkel des Stabs
%      x4 = Winkelgeschwindigkeit des Stabs

```

```

% hAxes := handle des Achsensystems, in das gezeichnet werden soll
9 %      (Leere Matrix ([]), wenn ein neues Figure erstellt
%      werden soll.)
% p      := "Pause" // Zeit zwischen einzelnen gezeichneten Zuständen
%      Standardwert: p = 0.025
% rec    := Video aus sim?
14 %      rec == 1 --> aufnehmen
%      rec == 0 --> NICHT aufnehmen
%      Standardwert: rec = 0

    if isempty(hAxes)
19        figure();
        hAxes = axes();
    else
        cla(hAxes);
    end

24    if (nargin < 5)
        p = 0.025;
        rec = 0;
    elseif (nargin < 6)
29        rec = 0;
    end

    if (rec == 0)
        vFrames = [];
34    end

    % Es werden nur Zustand 1 und 3 benötigt,
    % zur besseren Lesbarkeit werden diese in neue
    % Variablen "kopiert"
39    x1roh = mX(1:end,1);
    x3roh = mX(1:end,3);

    % Resampling
    vTroh = vT;
44    vT = 0:p:vTroh(end);
    x1 = interp1( vTroh, x1roh, vT );
    x3 = interp1( vTroh, x3roh, vT );

    % Berechnen der Stabposition für alle Zeitpunkte
49    l = stPendel.lPendel;
    stab = struct();
    for i=1:length(vT)
        stab.AnfangX(i) = x1(i) + 0;
        stab.AnfangY(i) = 0 + 0;
54        stab.EndeX(i) = x1(i) + sin(x3(i)) * l;
        stab.EndeY(i) = 0 - cos(x3(i)) * l;
    end

```

```

59 % für Axenskalierung
maxX=max(abs(stab.EndeX));
maxY=max(abs(stab.EndeY));

maxRange=1.1*max(maxX, maxY);

64 % "aufrufen" des übergebenen/erzeugten Axensystems + Skallierung
axes(hAxes)
axis(maxRange*[-1 1 -1 1]);
% WICHTIG: x- bzw y-Achsen müssen gleich skaliert werden, da sonst
%           das Pendel beim Schwingen verzerrt!

69 % Achsenbeschriftung
xlabel('x [m]');
ylabel('y [m]');

74 % Zeichnen der Anfangslage
line([stab.AnfangX(1) stab.EndeX(1)], [stab.AnfangY(1) stab.EndeY(1)], '→'
    'LineWidth',1, 'Color',[0 0 0]);
hold on;

% Zeichnen der "Tischlinie"
79 line(maxRange*[-1 1], [0 0], 'LineWidth',1, 'Color','b', 'LineStyle','--')→
    ←;

% Anfangslage Schlitten zeichnen
b = stPendel.bSchlitten;
h = stPendel.hSchlitten;
84 rectangle('Position',[stab.AnfangX(1)-b/2, stab.AnfangY(1)-h/2, b, h])→
    ←;

% handle des aktiven figures für vid
hFig = gcf();

89 % Berechnen wann ein Frame gespeichert wird (auf 500 Frames →
    ←beschränkt)
% (nur relevant, wenn rec == 1)
fr = ceil(length(vT)/500);
n = 1;

94 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Zeichnen der Zustände des Pendels
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i=1:length(vT)
    if i>1
99         delete(hLine);
        delete(hRec);
        end

        hRec = plotSchlitten(stab.AnfangX(i), stab.AnfangY(i), b, h);

```

```

104     hLine = line([stab.AnfangX(i) stab.EndeX(i)], [stab.AnfangY(i) stab→
        ↵.EndeY(i)], 'LineWidth', 2, 'Color', [1 0 0]);

    plot(stab.EndeX(i), stab.EndeY(i), 'r');

    % Zeit im Titel des Figures anzeigen
    % (auf 2Dezimalstellen gerundet)
109     title(['t⊥=⊥' num2str(round(100*vT(i))/100) '⊥sec']);

    % Aufnehmen, falls gewünscht
    % Die Anzahl der Frames wird auf 500 beschränkt
114     if ( rec && (mod(i-1, fr) == 0) )
        vFrames(n) = getframe(hFig);
        n = n + 1;
        % sollten Frames gespeichert werden ist keine zusätzliche
        % Pause nötig
119     else
        pause(p);
        % Pause, falls keine Frames gespeichert werden.
    end
    end % for i=1:length(vT)
124

    hold off; % nicht unbedingt notwendig, aber sauberer

end % function animiere Pendel

129
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Subfunction, die ein graues Rechteck zeichnet
% "Soll Schlitten darstellen"
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
134
function hRec = plotSchlitten(x,y,b,h)
    % x/y = Koordinaten des Mittelpunkts
    % b = Breite
    % h = Höhe
139

    vX = [x-b/2 x-b/2 x+b/2 x+b/2];
    vY = [y-h/2 y+h/2 y+h/2 y-h/2];

    hRec = fill(vX, vY, [0.9 0.9 0.9], 'LineStyle', 'none');
144

end % subfunction plotSchlitten

```

6 Protokoll

Im Protokoll soll hauptsächlich anhand ausgewählter Beispiele der Einfluss der Gewichtungsmatrizen \mathbf{Q} in der speziellen Form

$$\mathbf{Q} = \begin{bmatrix} q_1 & 0 & 0 & 0 \\ 0 & q_2 & 0 & 0 \\ 0 & 0 & q_3 & 0 \\ 0 & 0 & 0 & q_4 \end{bmatrix}$$

und

$$\mathbf{R} = R$$

verdeutlicht werden.

Wenn \mathbf{Q} wie hier nur Werte ungleich Null auf der Hauptdiagonalen besitzt, lassen sich die einzelnen Gewichtungsfaktoren q_i genau den einzelnen Zuständen x_i zuordnen. Multipliziert man denn Ausdruck $\mathbf{x}^T \mathbf{Q} \mathbf{x}$ im Gütemaß aus, ergibt sich nämlich

$$\mathbf{x}^T \mathbf{Q} \mathbf{x} = q_1 x_1^2 + q_2 x_2^2 + q_3 x_3^2 + q_4 x_4^2.$$

Um die Einflussmöglichkeiten über die Gewichtungen zu verdeutlichen, sind im Folgenden die Simulationsergebnisse für vier Fälle bei hängendem und für zwei Fälle bei stehendem Pendel gezeigt und kurz diskutiert. Eine Übersicht über die gewählten Gewichte ist in Tabelle 6.1 gezeigt.

Tabelle 6.1: Übersicht LQR

Simulation	q_1	q_2	q_3	q_4	R	siehe Abbildung
Regelung um unteren Arbeitspunkt						
1a	1	1	1	1	1	6.1
1b	1000	1	1	1	1	6.2
1c	1	1	1000	1	1	6.3
1d	1000	1	1	1	10	6.4
Regelung um oberen Arbeitspunkt						
2a	1000	1	1	1	1	6.5
2b	1	1	1000	1	1	6.6

In allen Fällen wurde eine Positionsänderung des Schlittens um 0,5 m betrachtet. Da die in Simulink implementierte Regelung das Schlitten-Pendel-System immer in die Ruhelage $\mathbf{x}_0 = [0 \ 0 \ 0 \ 0]^T$ (bzw. $\mathbf{x}_0 = [0 \ 0 \ \pi \ 0]^T$ bei stehendem Pendel) führt, wird dies so realisiert, dass der Anfangswert $\mathbf{x}(t=0) = [0,5 \ 0 \ 0 \ 0]^T$ bzw. $\mathbf{x}(t=0) = [0,5 \ 0 \ \pi \ 0]^T$ beträgt.

Als Ergebnis sind in Abbildung 6.1 bis 6.6 sind jeweils die Verläufe der Eingangsgröße $u(t)$ sowie die Schlittenposition $x(t)$ und der Pendelwinkel $\varphi(t)$ dargestellt. Man beachte dabei insbesondere für φ die unterschiedlichen Skalierungen! (Die Graphen wurden mit der Funktion `plotResults` erstellt, die in Listing 6.1 abgedruckt ist.)

Regelung um unteren Arbeitspunkt

In Simulation 1a wurde $\mathbf{Q} = \text{diag}([1 \ 1 \ 1 \ 1])$ (d. h. $q_1 = q_2 = q_3 = q_4 = 1$) und $R = 1$ gewählt. Es zeigt sich eine relativ langsame Regelung, die den Schlitten nach etwa 10 Sekunden in die Position $x = 0$ bringt, das Pendel schwingt dabei aufgrund der langsamen Bewegung des Schlittens nur sehr wenig (Amplitude maximal 0,005 rad).

Auch wenn in diesem Fall alle Gewichte einheitlich den Wert 1 besitzen, ist es aus physikalischer Sicht nicht unbedingt sinnvoll von einer Gleichgewichtung der Abweichungen zu sprechen, da hier nicht betrachtet wird in welcher Größenordnung denn die typischen Abweichungen jedes Zustandes und der Eingangsgröße liegen.

In Simulation 1b (Abbildung 6.2) wurde mit $\mathbf{Q} = \text{diag}([1000 \ 1 \ 1 \ 1])$ Abweichungen des Weges x deutlich stärker gewichtet. Der Schlitten wird jetzt sehr schnell in die Position $x = 0$ geführt (innerhalb 2 Sekunden), dafür wird das Pendel durch die hohe Beschleunigung am Anfang der Bewegung stark angeregt und schwingt mit Amplituden von maximal etwa 0,2 rad (ca. 11°), die nach 10 Sekunden jedoch wieder deutlich abgeklungen sind.

Als Gegenbeispiel wurde in Simulation 1c (Abbildung 6.3) die Abweichung des Pendelwinkels durch $\mathbf{Q} = \text{diag}([1 \ 1 \ 1000 \ 1])$ stark gewichtet. Entsprechend zeigt sich, dass die Pendelschwingungen effektiv unterdrückt werden. Allerdings wird dies dadurch erkauft, dass es nun wieder etwa 10 Sekunden dauert, bis der Schlitten die Position $x = 0$ erreicht hat.

Zuletzt wird der Einfluss der Gewichtung R der Eingangsgröße u betrachtet. Dabei werden die Zustände \mathbf{x} wie in Simulation 1b mit $\mathbf{Q} = \text{diag}([1000 \ 1 \ 1 \ 1])$ gewichtet, aber die Eingangsgröße mit $R = 10$. Vergleicht man nun Abbildung 6.4 mit Abbildung 6.2 kann man erkennen, dass durch die neue Gewichtung der maximale Wert (Betrag) der Eingangsgröße nur noch ein Drittel der aus Simulation 1b beträgt (-5 im Vergleich zu -15). Dadurch verlängert sich die Zeit, die der Schlitten benötigt um $x = 0$ zu erreichen auf etwa 3 Sekunden. Durch die geringere Beschleunigung halbieren sich auch ungefähr die Schwingungsamplituden des Pendels.

Regelung um oberen Arbeitspunkt

In Simulation 2a (Abbildung 6.5) wurde $\mathbf{Q} = \text{diag}([1000 \ 1 \ 1 \ 1])$ und $R = 1$ und in Simulation 2b (Abbildung 6.6) wurde $\mathbf{Q} = \text{diag}([1 \ 1 \ 1000 \ 1])$ und $R = 1$ gewählt. In beiden Fällen fällt auf, dass der Eingang u zu Beginn einen Impuls in die Gegenrichtung gibt, die dazu führt, dass sich der Wagen etwas in die Gegenrichtung bewegt, was allerdings nur bei größerem Zoom zu erkennen wäre. Das System hat also ein Allpassverhalten. Durch diese Wagenbewegung kippt das Pendel etwas in die Richtung des Punktes $x = 0$. Dann wird das geneigte Pendel zum Punkt $x = 0$ „gedrückt“.

Wenn das Hauptgewicht auf einen Lagefehler des Schlittens gelegt wird, dann überfährt dieser erstmals nach 2 Sekunden die Position $x = 0$ und führt eine gedämpfte, schwingende Bewegung aus. Wird die Winkelabweichung stark bestraft, dann bewegt sich der Wagen deutlich langsamer und erreicht erst nach 8 Sekunden erstmals $x = 0$, überschwingt dann aber auch nur weniger. Die Pendelbewegung ist minimal.

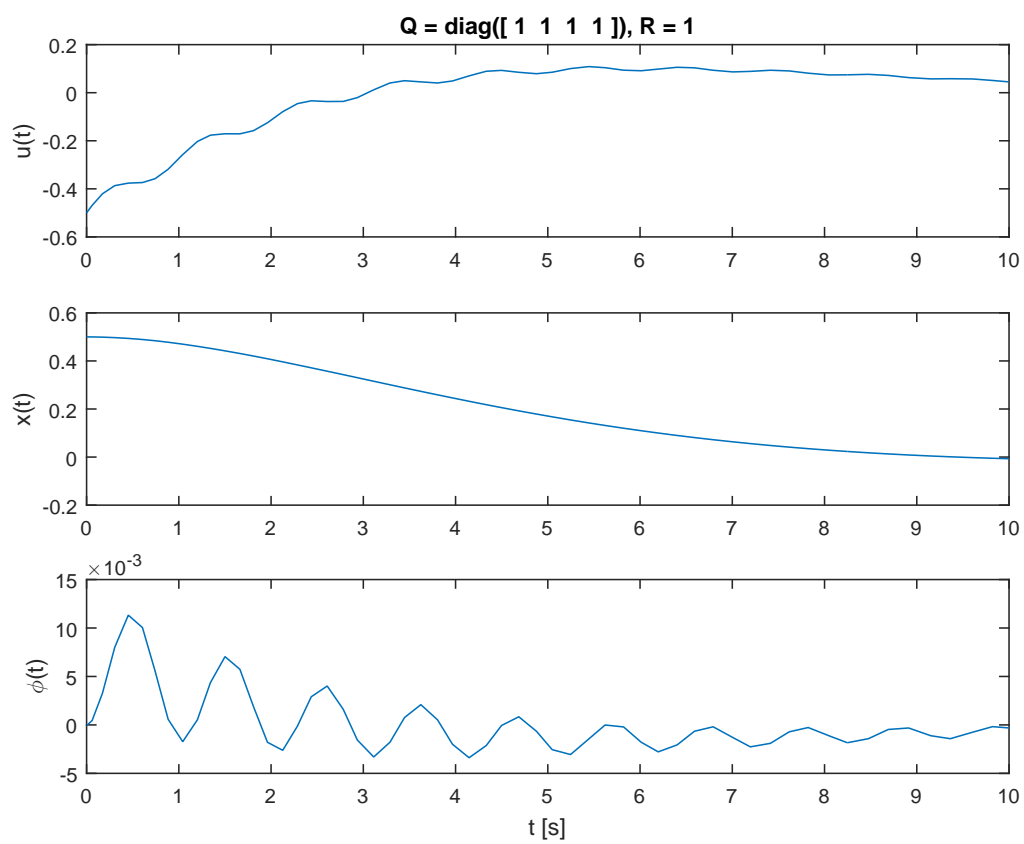


Abbildung 6.1: Simulation 1a: Hängendes Pendel, $Q = \text{diag}([1 \ 1 \ 1 \ 1]), R = 1$

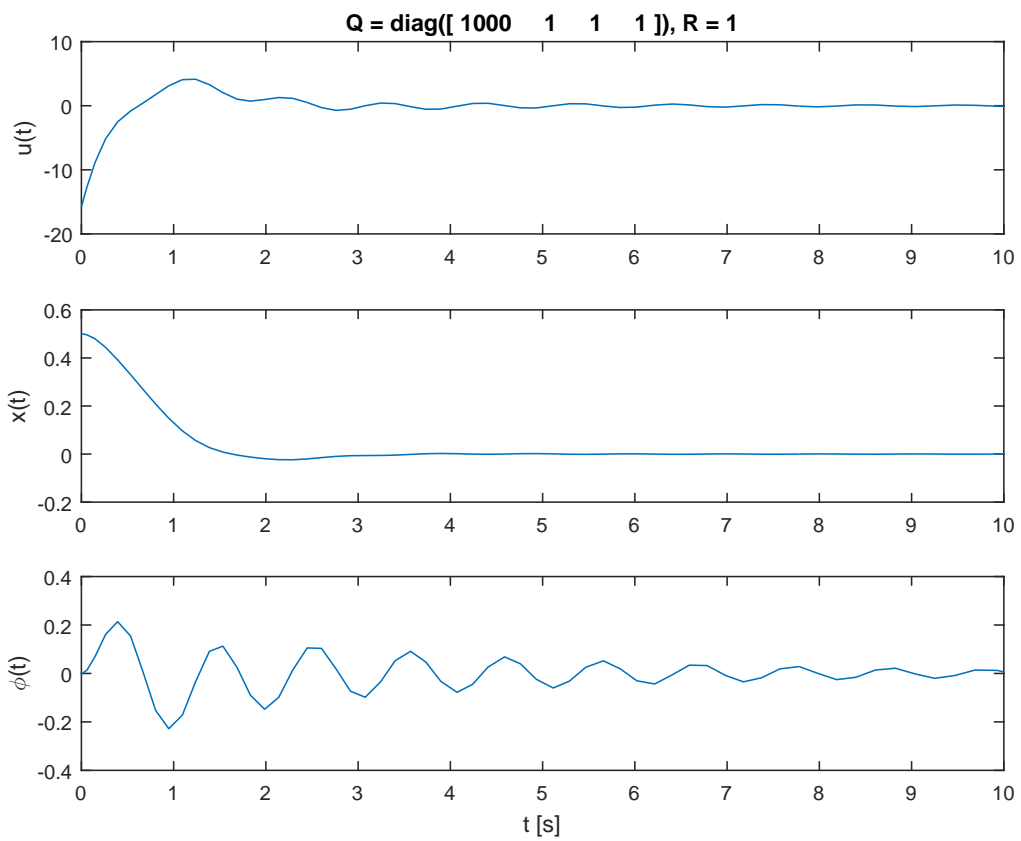


Abbildung 6.2: Simulation 1b: Hängendes Pendel, $Q = \text{diag}([1000 \ 1 \ 1 \ 1]), R = 1$

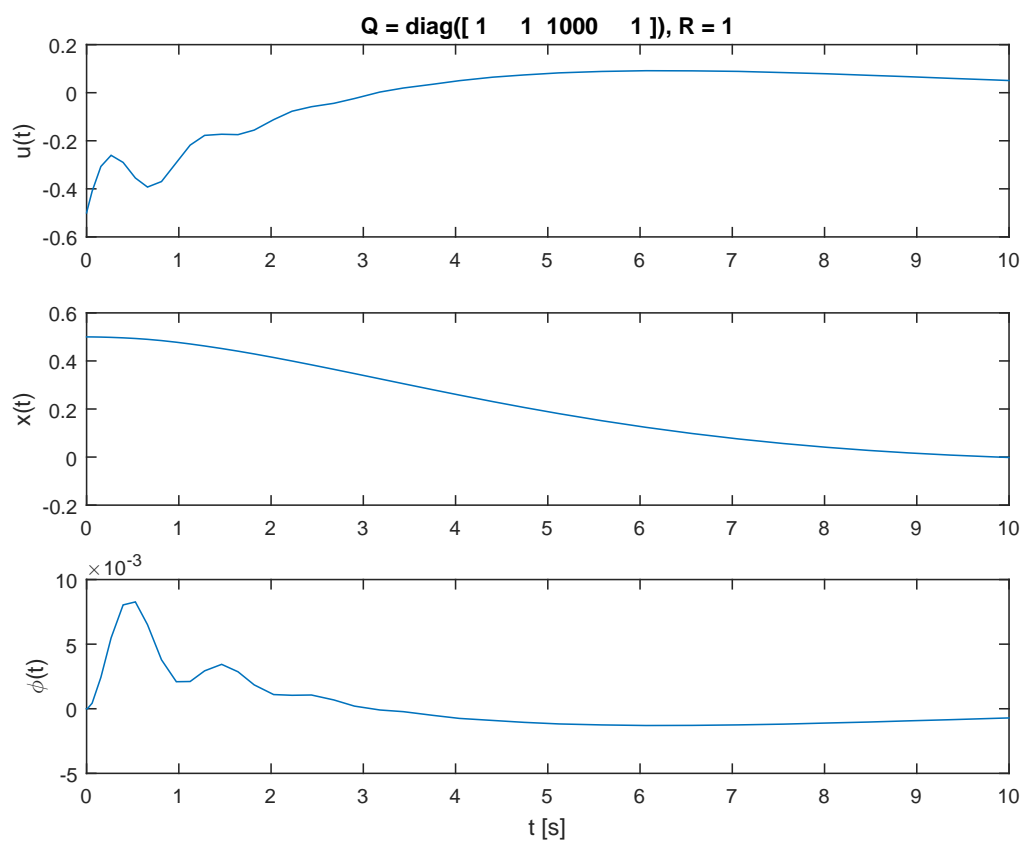


Abbildung 6.3: Simulation 1c: Hängendes Pendel, $Q = \text{diag}([1 \ 1 \ 1000 \ 1]), R = 1$

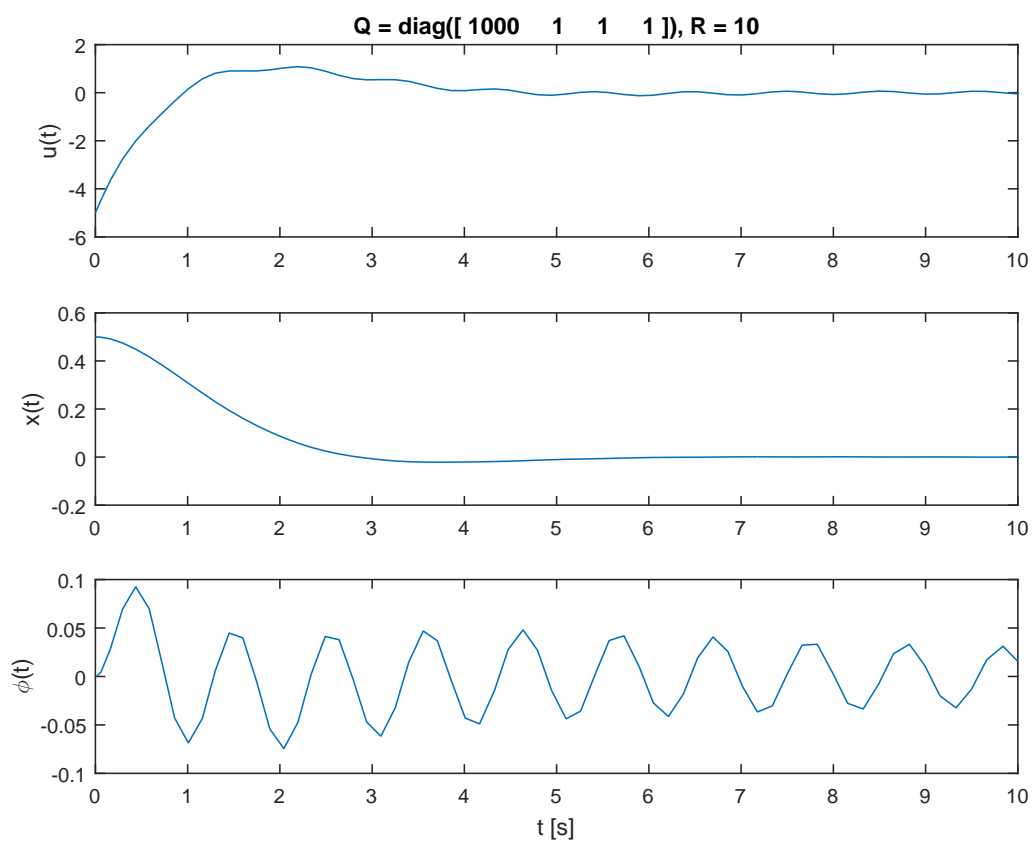


Abbildung 6.4: Simulation 1d: Hängendes Pendel, $Q = \text{diag}([1000 \ 1 \ 1 \ 1]), R = 10$

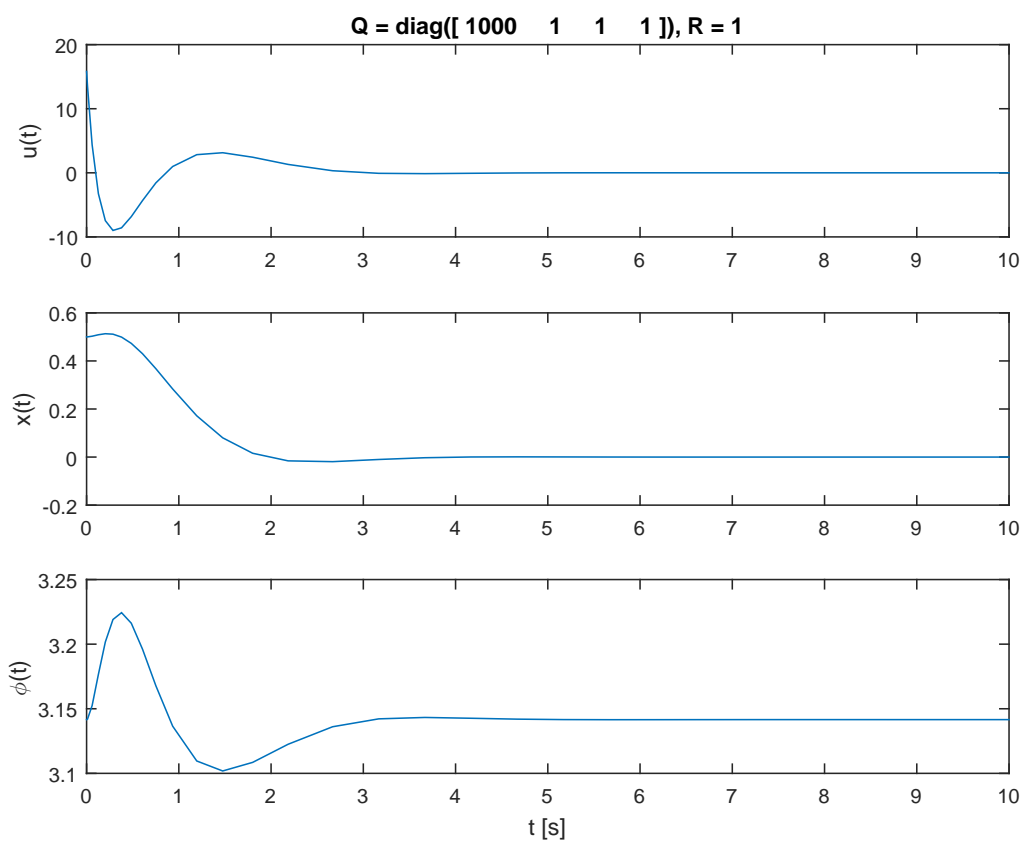


Abbildung 6.5: Simulation 2a: Stehendes Pendel, $Q = \text{diag}([1000 \ 1 \ 1 \ 1]), R = 1$

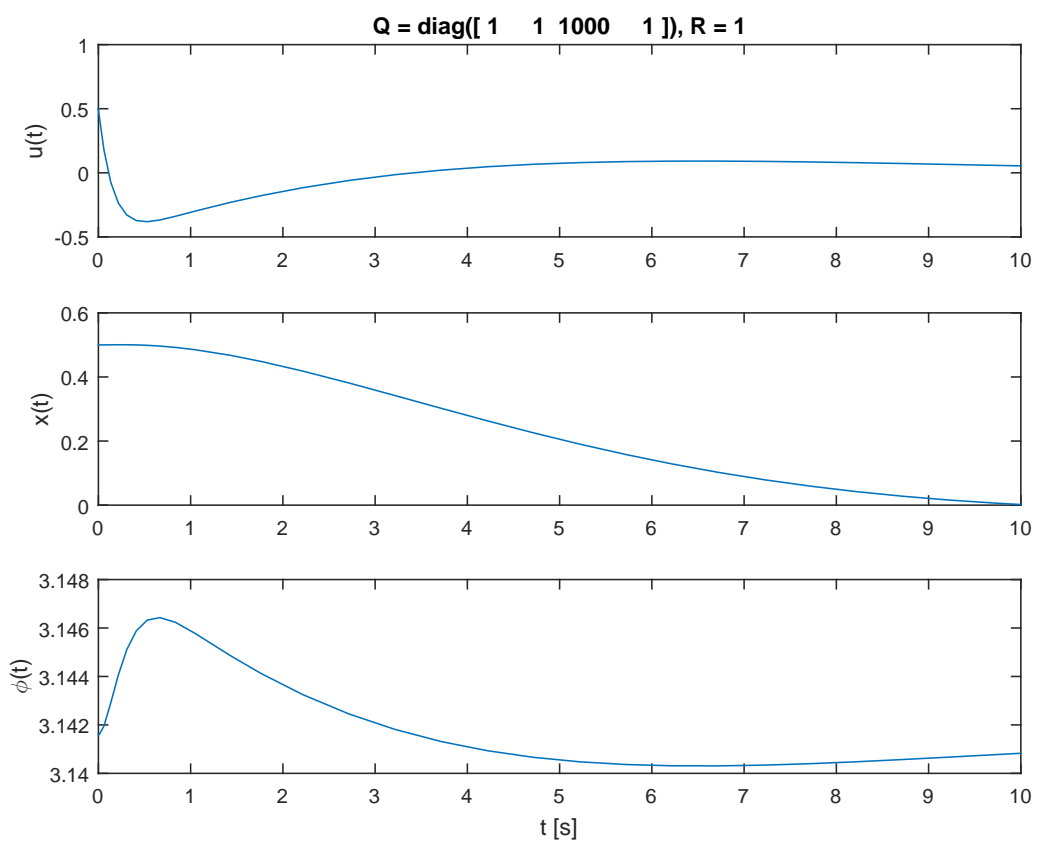


Abbildung 6.6: Simulation 2b: Stehendes Pendel, $Q = \text{diag}([1 \ 1 \ 1000 \ 1])$, $R = 1$

Listing 6.1: plotResults

```
function plotResults(vT, mX, vU, Q, R)

    figure();

5   vhAxes(1) = subplot(3, 1, 1);
    vhAxes(2) = subplot(3, 1, 2);
    vhAxes(3) = subplot(3, 1, 3);

    axes(vhAxes(1));
10   plot(vT, vU);
    title(['Q=diag([' num2str(diag(Q).') ']), R= num2str(R)]);
    ylabel('u(t)');

    axes(vhAxes(2));
15   plot(vT, mX(:,1));
    ylabel('x(t)');

    axes(vhAxes(3));
    plot(vT, mX(:,3));
20   ylabel('\phi(t)');
    xlabel('t[s]');

    linkaxes(vhAxes, 'x');

25 end % function plotResults
```

7 s-Function systemPendel_V3

Listing 7.1: systemPendel_V3

```
function systemPendel_V3(block)

    setup(block);

5 end

% *****
% Initialisierung
% *****
10 function setup(block)

    % Anzahl der Ein-/Ausgänge
    block.NumInputPorts = 1;
15 block.NumOutputPorts = 1;

    % Eigenschaften des Eingangs
    block.InputPort(1).Dimensions = 1;
    block.InputPort(1).DatatypeID = 0; % double
20 block.InputPort(1).Complexity = 'Real';
    block.InputPort(1).DirectFeedthrough = false;
    block.InputPort(1).SamplingMode = 'Sample';

    % Eigenschaften des 1. Ausgangs
25 block.OutputPort(1).Dimensions = 4;
    block.OutputPort(1).DatatypeID = 0; % double
    block.OutputPort(1).Complexity = 'Real';
    block.OutputPort(1).SamplingMode = 'Sample';

30 % Anzahl der Zustände
    block.NumContStates = 4;

    % Anzahl der Parameter
    block.NumDialogPrms = 2;
35

    % Abtastzeit definieren -> zeitkontinuierlich
    block.SampleTimes = [0 0];

40

    % weitere Methoden registrieren
```

```

    block.RegBlockMethod('InitializeConditions', @InitializeConditions);
    block.RegBlockMethod('Outputs', @Outputs);
45    block.RegBlockMethod('Derivatives', @Derivatives);
    block.RegBlockMethod('Terminate', @Terminate);

end

50
% *****
% Anfangsbedingungen setzen
% *****
function InitializeConditions(block)

55
    x0 = block.DialogPrm(2).Data;
    block.ContStates.Data = x0;

end

60
% *****
% Ausgänge berechnen
% *****
65 function Outputs(block)

    % Zustände ausgeben
    block.OutputPort(1).Data = block.ContStates.Data;

70 end

% *****
% Ableitungen berechnen
% *****
75 function Derivatives(block)

    % Parameter auslesen
    stPendel = block.DialogPrm(1).Data;
    ms = stPendel.mSchlitten;
80    mp = stPendel.mPendel;
    l = stPendel.lPendel;
    g = stPendel.g;

85    % Zustände auslesen
    x = block.ContStates.Data;
    xs = x(1);
    xs_d = x(2);
    phi = x(3);
90    phi_d = x(4);

    % Eingang auslesen

```

```

F = block.InputPort(1).Data(1);

95
% Ableitungen berechnen
phi_dd = ( ...
    (ms + mp) * (g * sin(phi)) + ...
    mp * phi_d^2 * 1/4 * sin(2*phi) + ...
100    cos(phi) * (F) ...
    ) / (mp * 1/2 * cos(phi)^2 - (ms + mp) * 2/3 * l);

xs_dd = -1/cos(phi) * ...
    (g * sin(phi) + 2/3 * l * phi_dd);
105

x_d = [ xs_d;
        xs_dd;
        phi_d;
        phi_dd];

110
% Ableitungen zuweisen
block.Derivatives.Data = x_d;

end

115

% *****
% Aufräumen (wenn nötig)
% *****
120 function Terminate(block)
end

```



Versuch 4

Beobachterentwurf – Benutzeroberflächen

8 Versuchsdurchführung	66
8.1 GUI 1. Teil	66
8.2 Beobachter	66
8.3 GUI 2. Teil	67
8.4 Einfluss des Beobachters	67
8.5 Protokoll	69
9 Programme	71
9.1 Aufgabe 1	71
9.2 Aufgabe 2	71
9.3 Aufgabe 3	72
9.4 Aufgabe 4	72
9.5 Aufgabe 5	73
9.6 Aufgabe 6	74
9.7 Aufgabe 7	78
10 Protokoll	81

8 Versuchsdurchführung

8.1 GUI 1. Teil

Aufgabe 4.1 (Durchführung/Nachbearbeitung):

- Kopieren Sie die gegebenen Dateien „simGUI.fig“ und „simGUI.m“ in ein Verzeichnis mit den im letzten Versuch erstellten Programmen.

Wenn Sie sich im letzten Versuch an die vorgegebenen Funktionsnamen und Syntax gehalten haben, dann sollte es möglich sein, die gegebene GUI zu starten und den Button „Berechne K“ zu betätigen.

Die berechneten Werte für K sollten dann im entsprechenden Feld angezeigt werden. Falls nicht, müssen Ihre Funktionen entsprechend angepasst werden.

Aufgabe 4.2 (Durchführung/Nachbearbeitung):

- Erweitern Sie die vorgegebene GUI in einem ersten Schritt dadurch, dass Sie neben den Reglerparametern auch die Pole des geschlossenen Regelkreises in einem Textfeld ausgeben.

Aufgabe 4.3 (Durchführung/Nachbearbeitung):

- Erweitern Sie die vorgegebene GUI in einem weiteren Schritt durch
 - Eingabefelder für die Anfangswerte des Systems und
 - Hinzufügen eines Buttons, durch den die Simulation ausgeführt wird. Die Animation des simulierten Systems soll in dem schon vorhandenen Achsensystem axes1 gezeigt werden.

Hinweis: Hierbei sollen natürlich die Funktionen aus Versuch 3 verwendet werden.

Aufgabe 4.4 (Durchführung/Nachbearbeitung):

- Fügen Sie einen Button STOP hinzu, mit dem Sie den Ablauf der Animation stoppen können.

Hinweis: Sie können hierzu eine globale Variable verwenden, die in der Callback-Funktion des Stop-Button gesetzt wird und in der for-Schleife abgefragt wird.

8.2 Beobachter

Aufgabe 4.5 (Durchführung/Nachbearbeitung):

- Schreiben Sie eine Funktion

```
function L = berechneBeobachter(A, C, poleBeobachter)
```

die den Beobachter nach vorgebbaren Polen auslegt.

Aufgabe 4.6 (Durchführung/Nachbearbeitung):

- Erweitern Sie Ihr Simulink-Modell aus Versuch 3, so dass alternativ (über eine Variable wählbar) die Regelung über Zustandsrückführung (Versuch 3) oder Ausgangsrückführung und Beobachter erfolgt. Dazu könnten Sie bspw. einen „Switch“-Block benutzen.
- Erweitern sie Ihre Funktion `runPendel` so, dass alternativ der Beobachter verwendet werden kann. Die neue Syntax der Funktion könnte so aussehen

`[vT, mX, mXobs] = runPendel(stPendel, AP, K, x0, stObs)`

wobei die Struktur `stObs` alle notwendigen Daten enthalten würde, um den Beobachter in Simulink benutzen zu können. Die Entscheidung, ob der Beobachter verwendet werden soll (und keine Zustandsrückführung) würde in diesem Fall durch die Anzahl der tatsächlich übergebenen Argumente (5 oder 4) getroffen werden.

Der Rückgabewert `mXobs` enthält die geschätzten Zustände.

- Testen Sie die neue Funktionalität zunächst ohne GUI.

Hinweise:

Wird der Beobachter verwendet, muss der Ausgang der s-Function transformiert werden, da dieser alle Systemzustände enthält. Dazu könnte entweder der Ausgang mit einer geeigneten Matrix multipliziert oder der Block „Selector“ aus der Gruppe „Signal Routing“ verwendet werden.

Die Funktion `runPendel` soll alle notwendigen Schritte zum Starten des Simulink-Modells beinhalten. D. h. diese Funktion sollte auch funktionieren, wenn vorher alle Variablen im Workspace gelöscht werden.

8.3 GUI 2. Teil

Aufgabe 4.7 (Durchführung/Nachbearbeitung):

- Erweitern Sie Ihre GUI um eine Auswahlmöglichkeit „Beobachter oder Zustandsrückführung“. Sehen Sie außerdem Eingabefelder für die Beobachtereigenwerte und die Anfangswerte für den Beobachter vor.
- Erweitern Sie die Callback-Funktion der Schaltfläche derart, dass je nach Wahl ggfs. die Beobachtermatrix `L` berechnet und die Funktion `runPendel` entsprechend aufgerufen wird.

In Abbildung 8.1 ist ein Beispiel gezeigt, wie die Oberfläche am Ende aussehen könnte.

8.4 Einfluss des Beobachters

Aufgabe 4.8 (Durchführung/Nachbearbeitung):

- Untersuchen Sie anhand von ein paar Beispielen und geeigneten Graphen den Einfluss des Beobachters und der Beobachtereigenwerte auf die geschätzten Zustände und das Verhalten des Regelkreises.

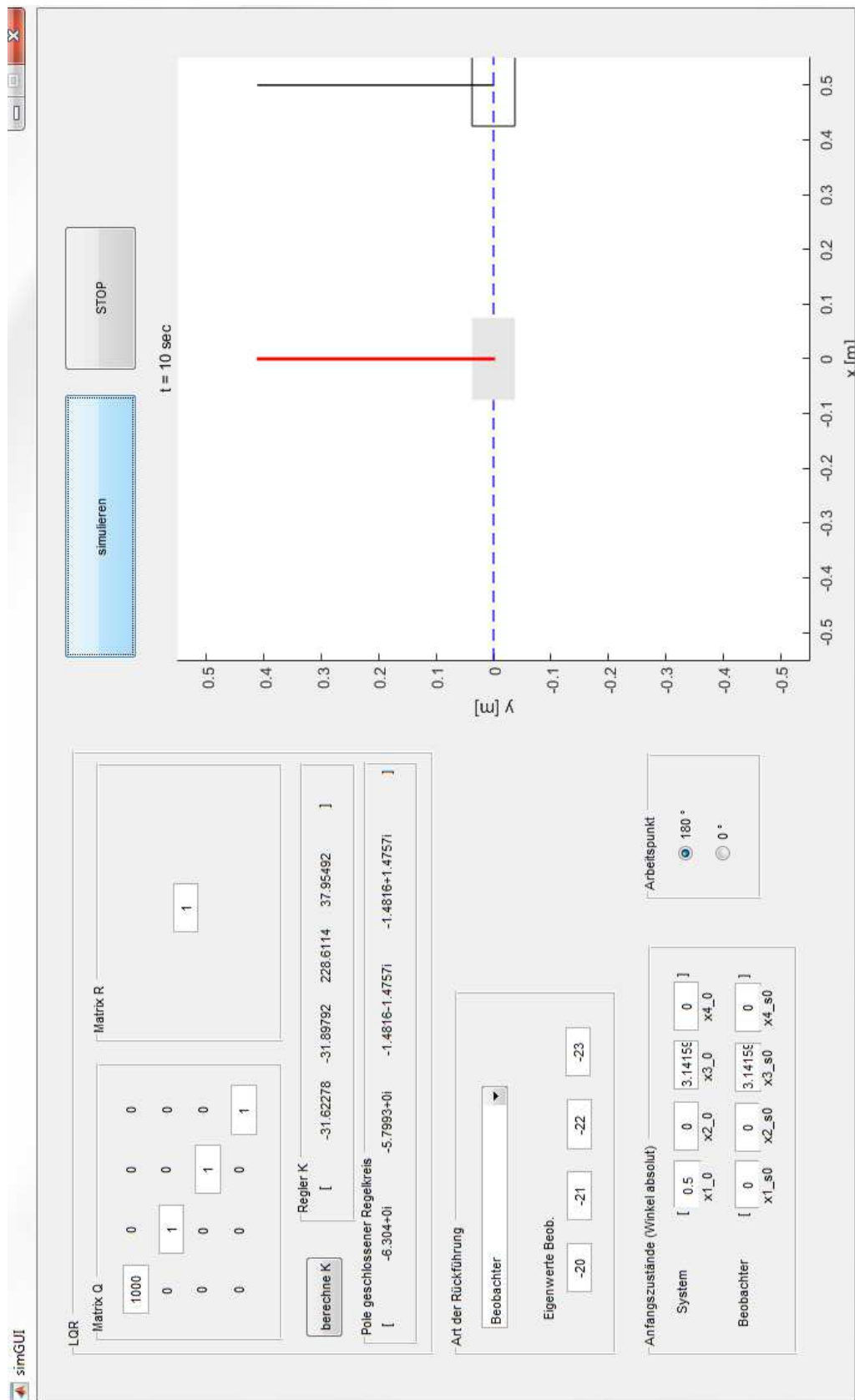


Abbildung 8.1: Mögliches Aussehen der GUI mit allen Funktionen

-
- Was fällt bei den Werten für die gemessenen Zustände x_1 und x_3 auf? Wie könnte das Verhalten verbessert werden?

8.5 Protokoll

Das Protokoll soll die Ergebnisse der Aufgabe 8 beinhalten.

Fügen Sie bitte außerdem

- alle Screenshots des Simulink-Modells mit Beobachter,
- sowie (unkommentiert) den Quellcode der relevanten Callback-Funktionen,
- sowie der Funktionen `berechneBeobachter` und `runPendel`

Ihrem Protokoll bei.



9 Programme

9.1 Aufgabe 1

Sollte mit der Lösung aus Versuch 3 sofort funktionieren.

Listing 9.1: Auszug 1 von berechneK_Callback

```
[...]  
  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
4 % Auslesen des Arbeitspunkts  
value = get(h.ap1, 'Value');  
if (value == 1)  
    arbeitspunkt = pi;  
else % (value == 0)  
9     arbeitspunkt = 0;  
end  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
  
[...]
```

9.2 Aufgabe 2

Dazu muss ein neues Textfeld erstellt werden. In dieser Musterlösung hat dieses Textfeld das Tag poleRK bekommen. Dann muss nur noch die Funktion berechneK_Callback entsprechend angepasst werden. (siehe Listing 9.2.)

Listing 9.2: Auszug 2 von berechneK_Callback

```
[...]  
  
2  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
  
pendeldataen = ladePendel();  
[A, B, C, D] = linPendelZR(pendeldataen, arbeitspunkt);  
7 [K, poleRK] = berechneLQR(A, B, C, D, Q, R);  
  
% Anzeigen des Vektors 'K' im Textfeld 'reglerK'  
set(h.reglerK, 'String', num2str(K));  
set(h.poleRK, 'String', num2str(poleRK.));  
12  
% end function berechneK_Callback
```

Die eckigen Klammern in der Oberfläche sind durch weitere Textfelder erzeugt und nicht notwendig. Diese Klammern könnten auch direkt in dem Textfeld poleRK erzeugt werden:

```
poleRKText = [ '[' num2str(poleRK.') ']' ];  
set(h.poleRK, 'String', poleRKText);
```

9.3 Aufgabe 3

Hierzu müssen vier Textfelder hinzugefügt werden, die in dieser Musterlösung die Tags x1_0, x2_0, x3_0 und x4_0 bekommen haben. Ebenso muss eine Schaltfläche erstellt werden (simulieren).

Die Ereignisfunktion simGUI_OpeningFcn der GUI kann dazu genutzt werden, beim Starten der GUI Anfangswerte einzutragen. In Listing 9.3 ist gezeigt.

Listing 9.3: simGUI_OpeningFcn mit Initialisierung der Felder mit den Anfangswerten des Systems

```
1 % --- Executes just before simGUI is made visible.  
function simGUI_OpeningFcn(hObject, eventdata, handles, varargin)  
% This function has no output args, see OutputFcn.  
% hObject    handle to figure  
% eventdata  reserved - to be defined in a future version of MATLAB  
6 % handles    structure with handles and user data (see GUIDATA)  
% varargin   command line arguments to simGUI (see VARARGIN)  
  
% Choose default command line output for simGUI  
handles.output = hObject;  
  
11 % Update handles structure  
guidata(hObject, handles);  
  
% UIWAIT makes simGUI wait for user response (see UIRESUME)  
16 % uiwait(handles.figure1);  
  
    set(handles.Q11, 'String', 100);  
    set(handles.Q22, 'String', 1);  
    set(handles.Q33, 'String', 100);  
21 set(handles.Q44, 'String', 1);  
    set(handles.R, 'String', 1);  
  
    set(handles.x1_0, 'String', 0.5);  
    set(handles.x2_0, 'String', 0);  
26 set(handles.x3_0, 'String', pi);  
    set(handles.x4_0, 'String', 0);
```

Die Callback-Funktion simulieren_Callback der neuen Schaltfläche ist unter Aufgabe 6 in Listing 9.8 zu sehen. (Mit den Erweiterungen für den Beobachter.)

9.4 Aufgabe 4

Es wurde eine Schaltfläche „Stop“ hinzugefügt. In der zugehörigen Callback-Funktion wird der globalen Variablen STOP der Wert 1 zugewiesen.

Listing 9.4: btnSTOP_Callback

```
function btnSTOP_Callback(hObject, eventdata, handles)
% hObject    handle to btnSTOP (see GCBO)
3 % eventdata reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

global STOP;
STOP = 1;
```

Die Funktion animierePendel wurde entsprechend erweitert.

Listing 9.5: Auszug aus animierePendel

```
function vFrames = animierePendel(vT, mX, pendelDaten, hAxes, p, rec)

3     [...]

    % globale Variable definieren und zurücksetzen
    global STOP;
    STOP = 0;

8     [...]

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Zeichnen der Zustände des Pendels
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
13    for i=1:length(vT)

        [...]

18        if STOP
            break;
        end

        [...]

23    end % for i=1:length(vT)

end % function animiere Pendel
```

9.5 Aufgabe 5

Hier wird die MATLAB-Funktion place verwendet, um die Beobachtermatrix **L** über das duale System als Zustandsregler zu entwerfen.

Listing 9.6: berechneBeobachter

```
function [L] = berechneBeobachter(A, C, obspoles)
```

```
4      L = place(A.', C, obspoles).';  
  
end % function berechneBeobachter
```

9.6 Aufgabe 6

Simulink-Modell

In Abbildung 9.1 ist das Simulink-Modell der Regelung mit Beobachter gezeigt. Wichtig sind dabei die folgenden Punkte:

- Es gelten die Hinweise zum Modell in der Musterlösung des dritten Versuchs.
- Es wird dieselbe s-Function wie in Versuch 3 benutzt.
- Die Einstellungen des „Selector“-Blocks sind in Abbildung 9.2 gezeigt.
- Alternativ könnte man auch eine Matrix-Multiplikation wie in Abbildung 9.3 gezeigt verwenden.
- Die Variable `mBeobachter` enthält (neben den anderen Zustandsgrößen) die Winkelwerte bezogen auf den linearisierten Arbeitspunkt. Dies muss später bei der Auswertung bedacht werden.
- Hier wird mit der Variablen `wahl` zwischen Zustandsrückführung `wahl == 1` und Beobachter `wahl == -1` umgeschaltet (Abbildung 9.1c).
- Im Integrator des Beobachters (Abbildung 9.1b) wird der Anfangswert mit der Variablen `x0_obs` festgelegt.

Funktion `runPendel`

Damit es übersichtlicher bleibt, wird für die Beobachterdaten eine Struktur mit den Elementen

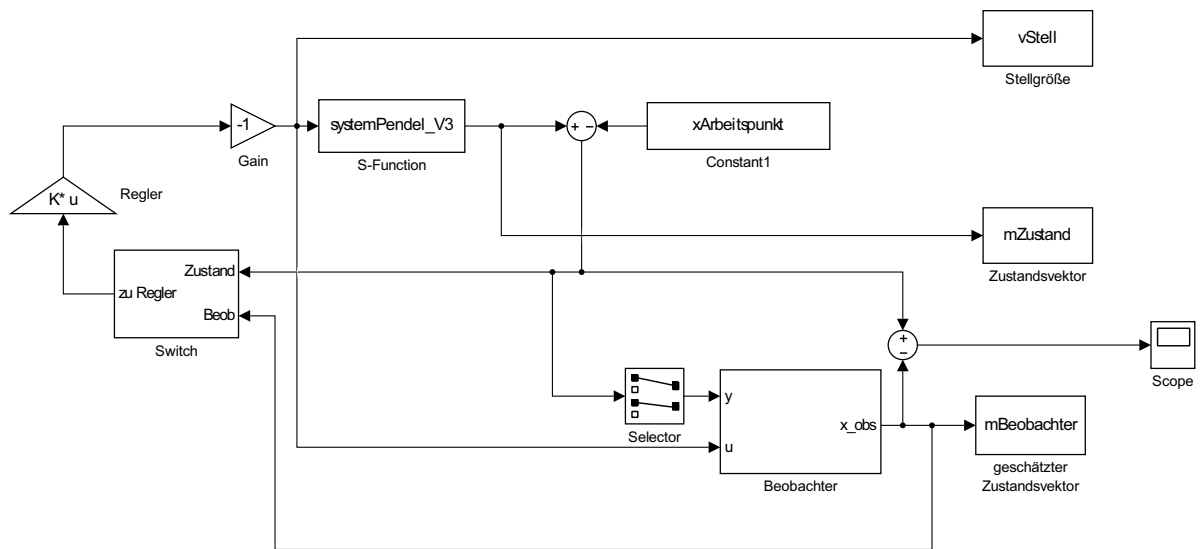
```
.x0  
.A  
.B  
.C  
.L
```

verwendet.

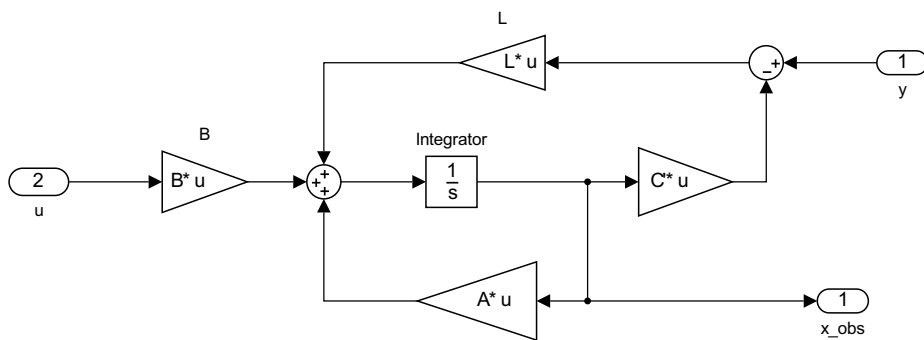
Wenn kein Beobachter verwendet werden soll, dann müssen trotzdem die Variablen `A`, `B`, `C`, `L` und `x0_obs` im Base-Workspace definiert sein. Daher werden hier in diesem Fall Null-Matrizen erzeugt. Es wird kein Beobachter verwendet, wenn entweder das letzte Argument `stObs` gar nicht übergeben wurde oder wenn eine leere Matrix übergeben wird.

Listing 9.7: `runPendel`

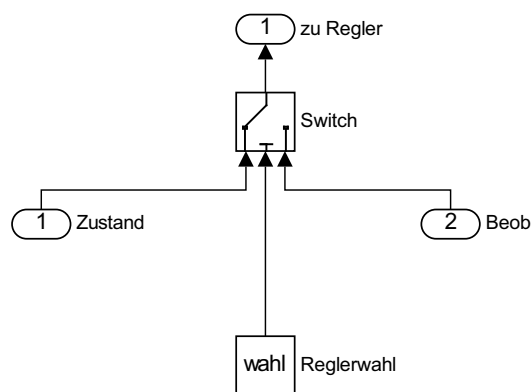
```
function [vT, mX, mB, vU] = runPendel(stPendel, AP, K, x0, stObs)  
  
    xArbeitspunkt = [0, 0, AP, 0];  
  
4    % Für Modell nötige Variablen im Base-Workspace anlegen
```



(a) Hauptmodul



(b) Submodel Beobachter



(c) Submodel Switch

Abbildung 9.1: Regelung mit Beobachter unter Simulink

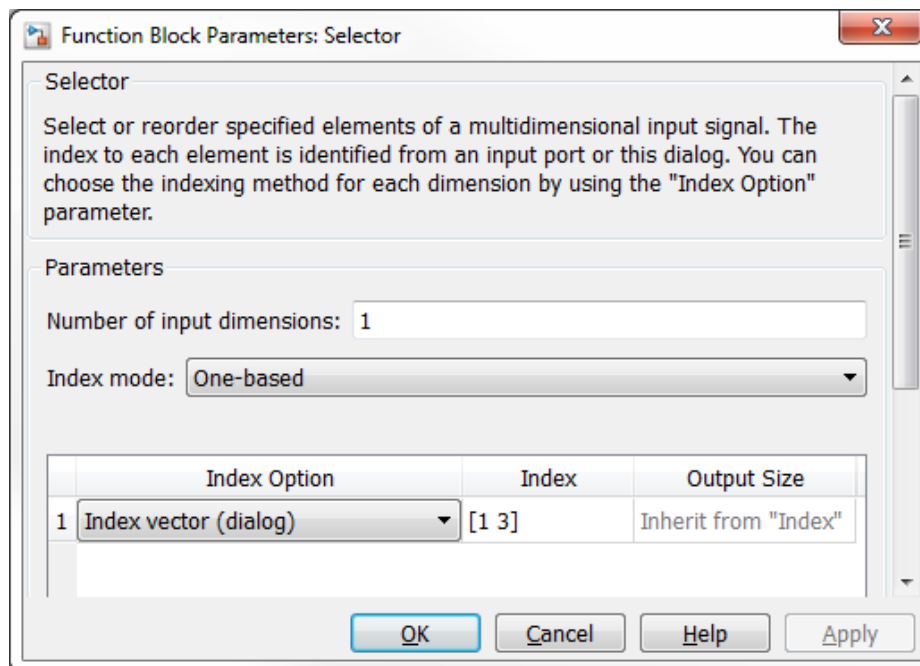


Abbildung 9.2: Einstellung des „Selector“-Blocks

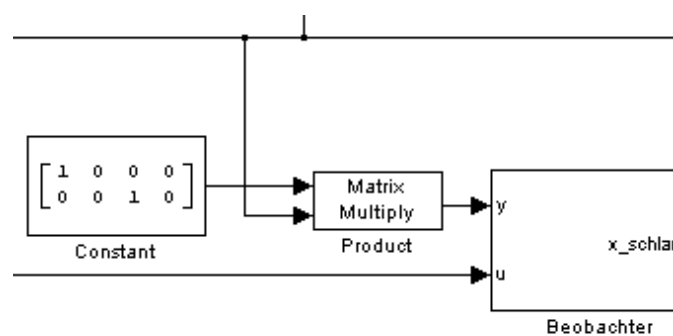


Abbildung 9.3: Variante bei Regelung mit Beobachter unter Simulink

```

assignin('base', 'xArbeitspunkt', xArbeitspunkt);

assignin('base', 'stPendel', stPendel);
9 assignin('base', 'x0', x0);
assignin('base', 'K', K);

% Wenn 5 Argumente übergeben wurden UND stObs nicht leer ist
% wird der Beobachter verwendet
14 if ((nargin == 5) && ~isempty(stObs))

    x0obs = stObs.x0;
    % In stObs.x0 wird Winkel absolut angegeben
    x0obs(3) = x0obs(3) - AP;

19
    assignin('base', 'x0_obs', x0obs);
    assignin('base', 'L', stObs.L);
    assignin('base', 'A', stObs.A);
    assignin('base', 'B', stObs.B);
24 assignin('base', 'C', stObs.C);

    assignin('base', 'wahl', -1);
else
    % Diese Variablen müssen auch dann im
    % Workspace definiert sein, wenn kein
    % Beobachter verwendet wird.
29 assignin('base', 'x0_obs', zeros(4,1));
    assignin('base', 'L', zeros(4,2));
    assignin('base', 'A', zeros(4,4));
34 assignin('base', 'B', zeros(4,1));
    assignin('base', 'C', zeros(4,2));

    assignin('base', 'wahl', 1);
end % if ((nargin == 5) && ~isempty(stObs))
39

% Simulation ausführen (hier mit fester Dauer von 10 Sekunden)
% Modellerte Zeitpunkte in "vT" speichern
vT = sim('Regelung_V4', 10);
44

% Modell schreibt "mZustand" in den Workspace.
% Dieses als "mX" zurückgeben.
mX = mZustand;
% Entsprechend "vInput" als "vU"
49 vU = vStell;
% und "mBeobachter" als "mB" zurückgeben.
mB = mBeobachter;
% mBeobachter enthält relativen Winkel
mB(:, 3) = mB(:, 3) + AP;
54

end % function runPendel

```

9.7 Aufgabe 7

In der Musterlösung erfolgt die Auswahl ob Beobachter oder Zustandsrückführung durch ein *Pop-up menu*.

Es ist auch ein Auswahlfeld für „Aufnahme“ vorgesehen. Wenn dieses ausgewählt ist, dann werden die Frames gespeichert und in den Base-Workspace kopiert. (Der Rahmen dieses Auswahlfeldes ist aber unsichtbar (Visible ist 'false') gemacht, und damit ist beim Ausführen das ganze Auswahlfeld nicht zu sehen.)

Listing 9.8: simulieren_Callback

```
% --- Executes on button press in simulieren.
function simulieren_Callback(hObject, eventdata, handles)
% hObject    handle to simulieren (see GCBO)
4 % eventdata reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

h = guihandles();

9 pendeldata = ladePendel();

14 % Arbeitspunkt auslesen
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
value = get(h.ap1, 'Value');          %%
if value == 1                          %% Auslesen des
    arbeitspunkt = pi;                %% Arbeitspunktes
19 elseif value == 0                  %%
    arbeitspunkt = 0;                 %%
end                                    %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

24 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
bRec = get(h.record_sim, 'Value');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
29 % Anfangswerte System auslesen

x1_0 = str2num(get(h.x1_0, 'String'));
x2_0 = str2num(get(h.x2_0, 'String'));
x3_0 = str2num(get(h.x3_0, 'String'));
34 x4_0 = str2num(get(h.x4_0, 'String'));

x0 = [x1_0 x2_0 x3_0 x4_0];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

39
```



```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Wahl der Rückführung
str = get(h.Regler_Wahl, 'String');
44 temp = get(h.Regler_Wahl, 'Value');

switch str{temp}
    case 'Beobachter'
        wahl = -1;
49    case 'Zustandsrückführung'
        wahl = 1;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

54 if (wahl == -1)
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Beobachter auslegen

    [A, B, C, D] = linPendelZR(pendeldata, arbeitspunkt);
59
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Anfangswerte Beobachter auslesen

    x1_s0 = str2num(get(h.x1_s0, 'String'));
64 x2_s0 = str2num(get(h.x2_s0, 'String'));
    x3_s0 = str2num(get(h.x3_s0, 'String'));
    x4_s0 = str2num(get(h.x4_s0, 'String'));

    x0_s = [x1_s0 x2_s0 x3_s0 x4_s0];
69 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    eig1 = str2num(get(h.eig1, 'String'));
    eig2 = str2num(get(h.eig2, 'String'));
    eig3 = str2num(get(h.eig3, 'String'));
74 eig4 = str2num(get(h.eig4, 'String'));

    eigBeob = [eig1 eig2 eig3 eig4];

    L = berechneBeobachter(A, C, eigBeob);
79

    stObs.A = A;
    stObs.B = B;
    stObs.C = C;
    stObs.L = L;
84 stObs.x0 = x0_s;
else % (wahl == 1)
    stObs = [];
end % if (wahl == -1) ... else ...
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
89

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Simulation
K = str2num(get(h.reglerK, 'String'));

94 [vT, mX, mXobs, vU] = runPendel(pendelraten, arbeitspunkt, K, x0, →
    ←stObs);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Animation
99 stFrames = animierePendel(vT, mX, pendelraten, h.axes1, 0.025, bRec);

if (bRec == 1)
    assignin('base', 'stFrames', stFrames);
end

104 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%if (wahl == -1)
%    plotResultsObs(vT, mX, mXobs, vU);
109 %end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% end function simulieren_Callback

```

10 Protokoll

Zum Untersuchen des Einflusses des Beobachters wird die Regelung um den unteren Arbeitspunkt betrachtet. (Für den oberen Arbeitspunkt ergibt sich qualitativ dasselbe, allerdings muss man da mehr darauf achten, dass der Regelkreis nicht instabil wird. Unten kann man einfacher extreme Werte ausprobieren.)

Es wird immer die Regelung von einer Anfangsposition $\mathbf{x}_0 = [0,5 \ 0 \ 0 \ 0]^T$ nach $\mathbf{0}$ betrachtet. Der verwendete LQ-Regler wurde mit den Gewichten $q_1 = q_2 = 100$ und $q_3 = q_4 = R = 1$ ausgelegt. Damit ergeben sich für den geschlossenen Regelkreis die Eigenwerte $\lambda_{\text{RK}} = [-0,8296 \pm 0,8253j, -0,4967 \pm 6,0558j]^T$.

Tabelle 10.1 zeigt die Übersicht über die Simulationen mit den jeweils verwendeten Eigenwerten des Beobachters sowie den Anfangswerten des Beobachters.

Tabelle 10.1: Übersicht über Simulationen mit Beobachter

Simulation	λ_{Obs}	\mathbf{x}_0^T	$\hat{\mathbf{x}}_0^T$	siehe Abbildung
1	[10 11 12 13]	[0,5 0 0 0]	[0 0 0 0]	10.1, 10.2
2	[1,0 1,1 1,2 1,3]	[0,5 0 0 0]	[0 0 0 0]	10.1, 10.3
3	[0,10 0,11 0,12 0,13]	[0,5 0 0 0]	[0 0 0 0]	10.1, 10.4
4	[0,10 0,11 0,12 0,13]	[0,5 0 0 0]	[0,5 0 0 0]	10.5
5	[1,0 1,1 1,2 1,3]	[0,5 0 0 0]	[0,5 1 0 0]	10.6

In den Simulationen 1 bis 3 war der Anfangswert des Beobachters immer $\hat{\mathbf{x}}_0 = [0 \ 0 \ 0 \ 0]^T$. In Abbildung 10.1 sind die sich ergebenden Verläufe für $x_1 = x$ (Position Schlitten) und $x_3 = \varphi$ (Winkel Pendel) dargestellt. Zum Vergleich sind auch die entsprechenden Verläufe, die sich bei der Zustandsrückführung ergeben dargestellt. Es ist gut zu erkennen, dass je langsamer die Beobachtereigenwerte sind, desto langsamer die Regelung der Schlittenposition wird.

Bei den schnellen Beobachtereigenwerten in Simulation 1 wird die Position $x = 0$ etwa nach der selben Zeit wie bei der Zustandsrückführung endgültig erreicht, allerdings schwingt der Schlitten vorher deutlich über. Auch ist eine deutliche Schwingung des Pendels zu erkennen. In Abbildung 10.2 ist zu sehen, dass die Abweichung des Anfangsschätzwertes für x_1 zu Beginn zu einer großen Abweichung des Schätzwertes für x_2 führt. So nimmt der Beobachter an, dass sich der Schlitten mit großer Geschwindigkeit nach rechts bewegt. Dies führt dazu, dass der Regler zu stark gegensteuert und dies erklärt das starke Überschwingen. Nach etwa 0,5 Sekunden stimmen die Schätzwerte mit den Istwerten der Zustände überein. (Die Winkelposition und -geschwindigkeit wird zu jeder Zeit korrekt geschätzt.)

Liegen die Beobachtereigenwerte in der Größenordnung der dominanten Eigenwerte (Simulation 2) wird die Endlage etwas später erreicht (hier nach etwa 8 Sekunden). Auch schwingt der Schlitten zunächst deutlich über die Position $x = 0$ hinaus. Die Schwingungen des Pendels sind vergleichbar mit denen bei der Zustandsrückführung. Für die geschätzten Werte (Abbildung 10.3) gilt qualitativ dasselbe was schon bei Simulation 1 festgestellt wurde, nur dass die fehlerhafte Schätzung für x_2 nicht so stark ist und dass es dafür mit 4 Sekunden länger dauert, bis die Zustände korrekt geschätzt werden.

Die ganz langsamen Eigenwerte des Beobachters in Simulation 3 führen zu einer sehr langsamen Regelung. Nach den 10 Sekunden, die in Abbildung 10.1 dargestellt sind, hat der Schlitten gerade einmal die Position $x = 0$ überfahren und führt eine sehr langsame Schwingung aus. Auch nach 30 Sekunden besteht noch eine Abweichung der geschätzten Zustände zu den Ist-Zuständen und die Endlage ist noch nicht erreicht, wie in Abbildung 10.4 zu erkennen ist. Auch zeigt sich hier, dass hier auch ein Schätzfeh-

ler für den Zustand x_3 entsteht.

Dass die Eigenwerte des Beobachters in diesem Fall nur einen Einfluss auf die Regelung nehmen, wenn eine Abweichung der geschätzten Zustände zu den Istzuständen besteht, lässt sich auch bei Simulation 4 (Abbildung 10.5) erkennen. Hier stimmen die Anfangswerte von Strecke und Beobachter genau überein, und es tritt damit auch im weiteren Verlauf nie eine Abweichung auf. Damit kann man sich den Beobachter auch „wegdenken“ und erhält eine Zustandsrückführung.

Im Normalfall lässt sich dies aber nicht ohne Weiteres verallgemeinern, da hier das linearisierte Modell sehr gut mit der Strecke übereinstimmt und kein Messrauschen oder andere Störungen auftreten, die trotz korrekter Anfangswerte zu einer Abweichung der Schätzwerte führen können.

Zuletzt wurde in Simulation 5 (Abbildung 10.6) ein Fall betrachtet, bei dem die Anfangsabweichung des Beobachters eine nicht gemessene Zustandsgröße betrifft. In diesem Fall führt eine Anfangsabweichung bei x_2 im weiteren Verlauf zu einer fehlerhaften Schätzung für x_1 (obwohl x_1 gemessen wird).

Letzteres führt auch auf den Vorteil eines reduzierten Beobachters. Da bei diesem die tatsächlich gemessenen Größen nicht mitgeschätzt werden, würden diese immer korrekt sein und dabei noch die Schätzwerte der anderen Zustände verbessern.

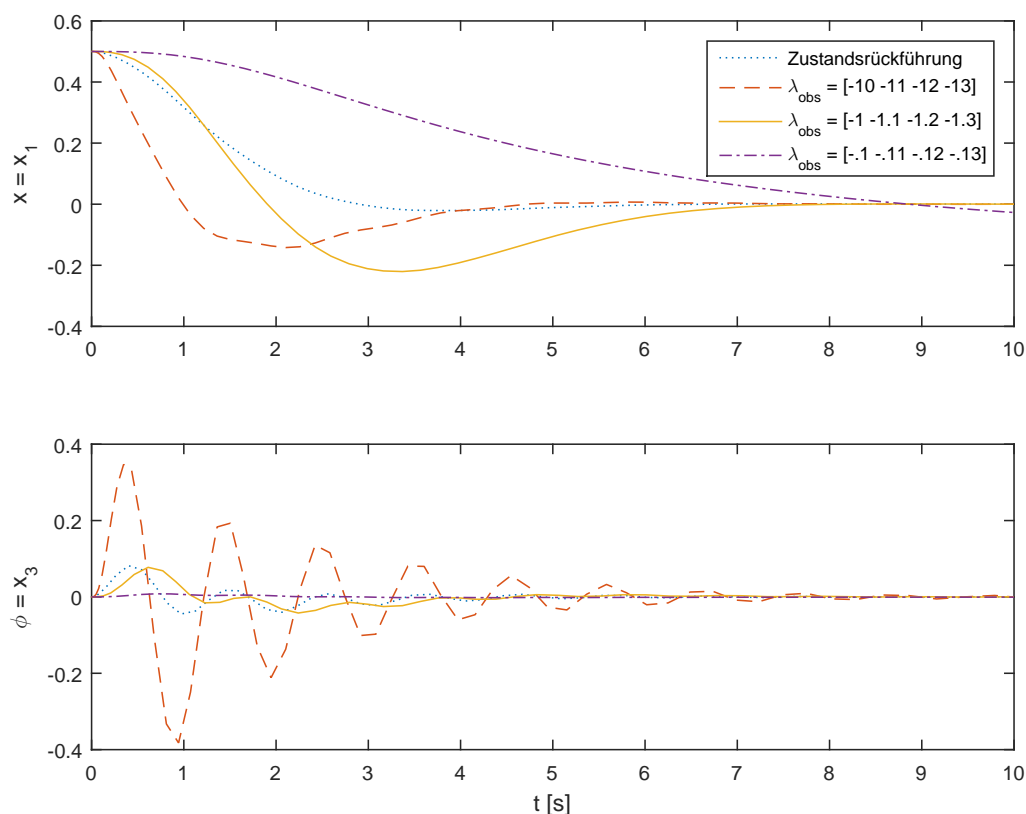


Abbildung 10.1: Verlauf von x_1 und x_3 bei verschiedenen Beobachtern ($x_0 - \hat{x}_0 = [0,5 \ 0 \ 0 \ 0]^T$)

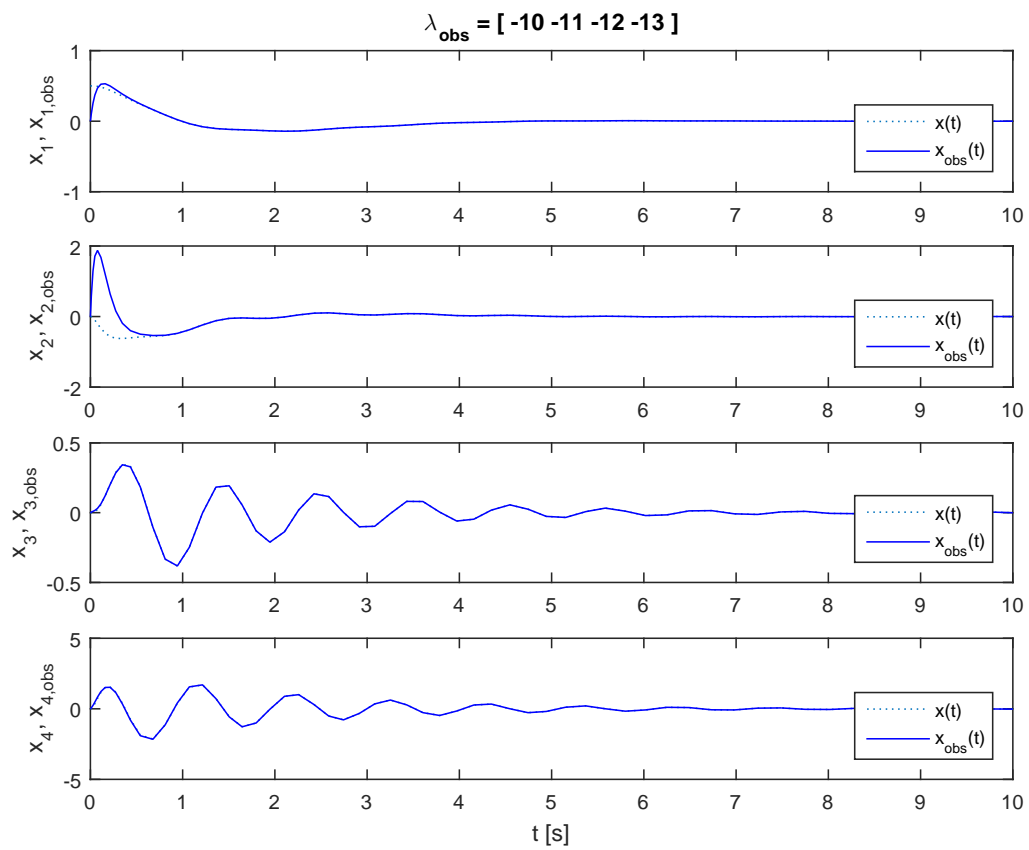


Abbildung 10.2: Verlauf von x und \hat{x} ($x_0 - \hat{x}_0 = [0,5 \ 0 \ 0 \ 0]^T$)

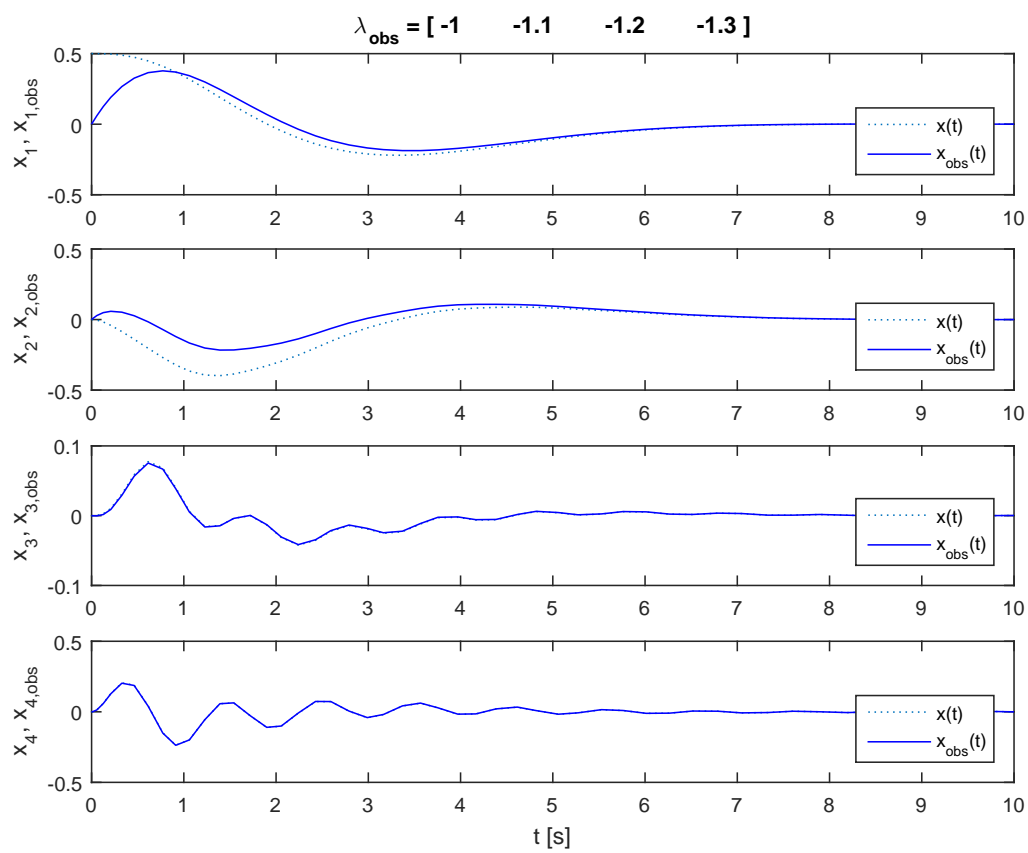


Abbildung 10.3: Verlauf von x und \hat{x} ($x_0 - \hat{x}_0 = [0,5 \ 0 \ 0 \ 0]^T$)

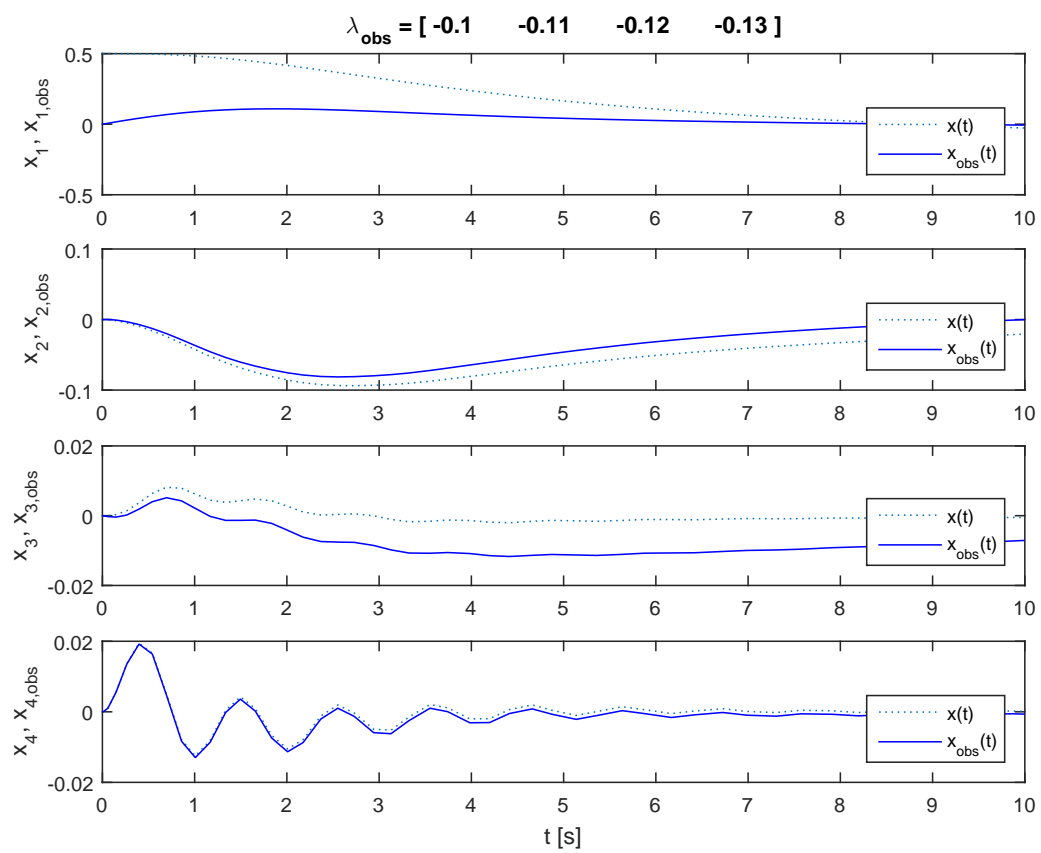


Abbildung 10.4: Verlauf von x und \hat{x} ($x_0 - \hat{x}_0 = [0,5 \ 0 \ 0 \ 0]^T$)

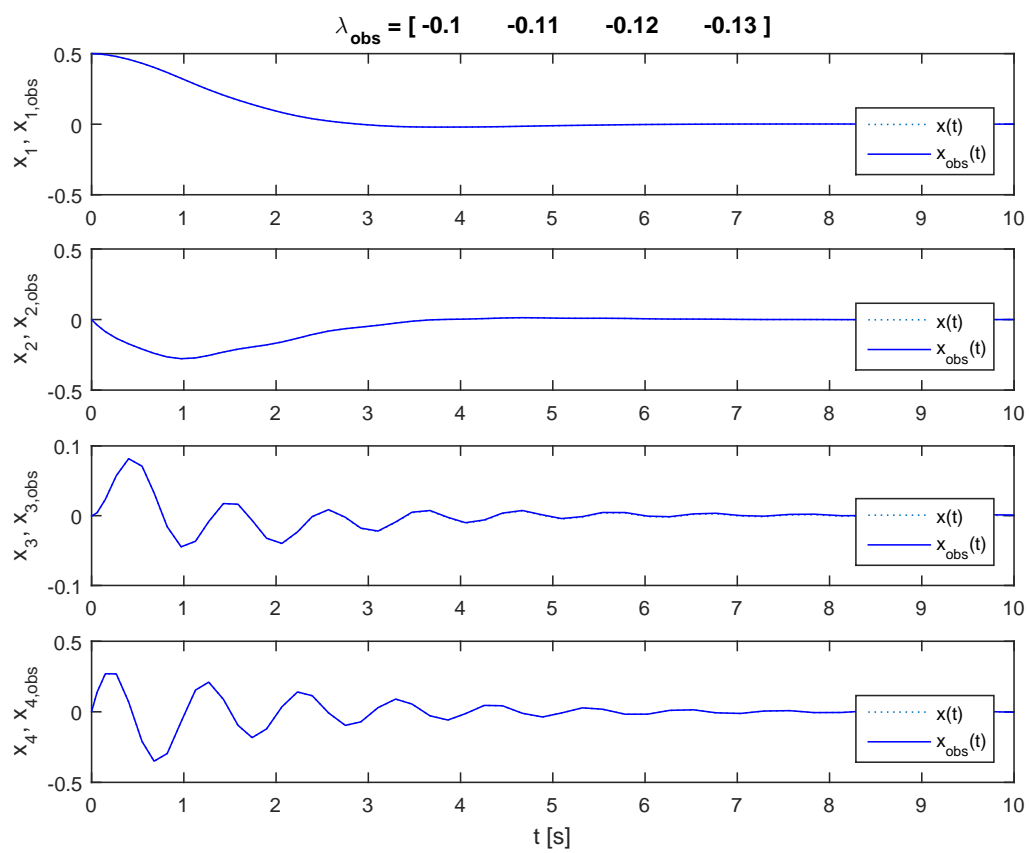


Abbildung 10.5: Verlauf von x und \hat{x} ($x_0 - \hat{x}_0 = [0 \ 0 \ 0 \ 0]^T$)

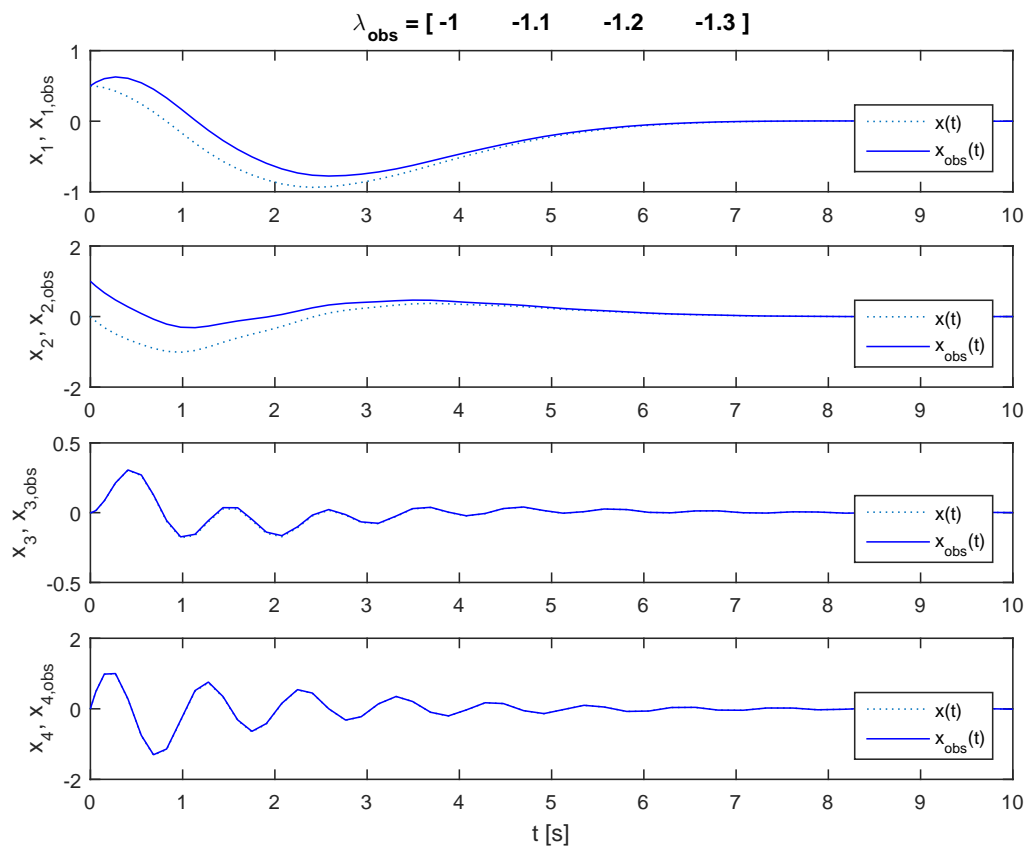


Abbildung 10.6: Verlauf von x und \hat{x} ($x_0 - \hat{x}_0 = [0 \ -1 \ 0 \ 0]^T$)



Versuch 5

Aufschwingsteuerung für das Pendel

11 Versuchsdurchführung	90
11.1 Berechnen der Trajektorien	90
11.2 Simulation	91
11.3 Protokoll	92
12 Programmierung	93
12.1 Berechnung der Trajektorien	93
12.2 Simulation	97
13 Auswertung	101
13.1 Berechnung der Trajektorien	101
13.2 Simulation	101
13.3 s-Function systemPendel_V5	110

11 Versuchsdurchführung

11.1 Berechnen von $u^*(t)$ und $x^*(t)$

11.1.1 Programmierung

Aufgabe 5.1 (Durchführung/Nachbearbeitung):

- Erstellen Sie eine Funktion

`stTraj = berechneTrajektorie(stPendel, T)`

die aus den Pendeldata (Struktur wie in Versuch 3 und 4) und der vorzugebenden Übergangszeit T die Verläufe der Eingangsgröße $u^*(t)$ und der Zustände $x^*(t)$ nach dem vorgestellten Verfahren mit `bvp4c` berechnet. Es bietet sich an, als Rückgabewert eine Struktur zu haben, in denen der Zeitvektor t mit der Steuerfolge u und den Zuständen x zurückgegeben werden. Außerdem könnte noch explizit die Übergangszeit T angegeben werden:

```
stTraj.T
      .vT
      .vU
      .mX
```

Um diese Funktion zu realisieren, können Sie sich an folgenden Schritten orientieren:

- Schreiben Sie eine Funktion `RandwertproblemDGL`, die das DGL-System (16.6) des Randwertproblems beschreibt:

`dx = RandwertproblemDGL(t, x, P, stPendel, T)`

Die Funktion hat als Eingangsparameter zum eine die von `bvp4c` geforderten Daten: Zeit t , Zustandsvektor x und den Vektor P (enthält die freien Parameter p_1 und p_2). Zudem sollen dieser Funktion auch die Struktur mit den Pendeldata und die gewünschte Übergangszeit T übergeben werden können. Da `bvp4c` nur die ersten drei Parameter benutzt, würde dies zum Absturz führen, d. h. es muss möglich sein, diese Funktion auch mit drei Parametern aufzurufen. Im Abschnitt 17.3 wurden dazu zwei Möglichkeiten vorgestellt, von denen Sie sich eine aussuchen können. Stimmen Sie die Funktion `RandwertproblemDGL` entsprechend auf Ihre Wahl ab.

- Schreiben Sie eine Funktion `RandwertproblemRB`:

`deltaRB = RandwertproblemRB(x0, xT, P)`

Sie gibt einen Spaltenvektor `deltaRB` mit der Abweichung der Werte bei $t = 0$ und $t = T$ zu den gegebenen Randwerten (16.7) (Auch wenn im vorliegenden Problem die Randwerte unabhängig von den Parametern p_1 und p_2 sind, muss der Parametervektor P als Argument angegeben werden, da dieser vom Löser übergeben wird.)

- Erstellen Sie in der Funktion `berechneTrajektorie` die Struktur `solinit` mit den Startwerten für den Löser. Geben Sie zunächst eine Aufteilung für t in 1000 Intervallen an. Für $y(t)$ geben Sie den linearen Verlauf der beiden Zustände von Anfangs- zum Endwert vor. Für p_1 und p_2 geben Sie jeweils Null vor.

-
- Lösen sie das Randwertproblem mit `bvp4c`.
 - Berechnen bzw. entnehmen Sie aus der von `bvp4c` zurückgegebenen Struktur den Verlauf für die Eingangsgröße $u^*(t)$ sowie den Verlauf aller vier Zustände $\mathbf{x}^*(t)$ und geben Sie diese in der oben angegebenen Struktur `stTraj` zurück.

11.1.2 Auswertung

Aufgabe 5.2 (Durchführung/Nachbearbeitung):

- Wählen Sie zum Testen Ihrer Funktion zunächst für T einen Wert zwischen 1,2 und 1,5 s.
- Betrachten Sie für verschiedene Übergangszeiten T die Trajektorie $y^*(t)$ und die Verläufe von $u^*(t)$ und $\mathbf{x}^*(t)$ und überprüfen Sie die Lösung. Plotten sie den Verlauf der Zustände $\mathbf{x}^*(t)$ und der Stellgröße $u^*(t)$.
- In welchem Wertebereich für T können Lösungen gefunden werden? Begründen sie über die Anschauung, ob das Nichtauffinden von Lösungen für große T am Löser oder an der Physik liegen wird.
- Versuchen Sie durch Variation der Übergangszeit T die Beschränkungen

$$|x| = 0,5 \text{ m}$$

$$|\dot{x}| = 1,6 \text{ m/s}$$

$$|\ddot{x}| = 12 \text{ m/s}^2$$

möglichst knapp einzuhalten. (x ist die Position des Schlittens.)

11.2 Simulation

Aufgabe 5.3 (Durchführung/Nachbearbeitung):

- Bauen Sie die Steuerung unter Simulink auf.

Benutzen Sie die s-Function `systemPendel_V5` als Modell für das System. Sie beschreibt das Schlitten-Pendel-System nach (16.2). (Im Unterschied zu `systemPendel_V3` wird hier als Eingang u die Beschleunigung des Schlittens und nicht die Kraft auf den Schlitten verwendet.) Als Parameter werden die Pendeldata und die Anfangsbedingung des Systems erwartet (siehe Abbildung 11.1). Der Ausgang dieser s-Function entspricht wie auch bei `systemPendel_V3` den Zuständen.

11.2.1 Auswertung

Aufgabe 5.4 (Durchführung/Nachbearbeitung):

- Vergleichen Sie anhand der Zeitplots (und der Animation) das Ergebnis der Simulation mit den theoretisch berechneten Trajektorien. Lassen Sie die Simulation etwa eine Sekunde länger als die Übergangszeit T laufen.

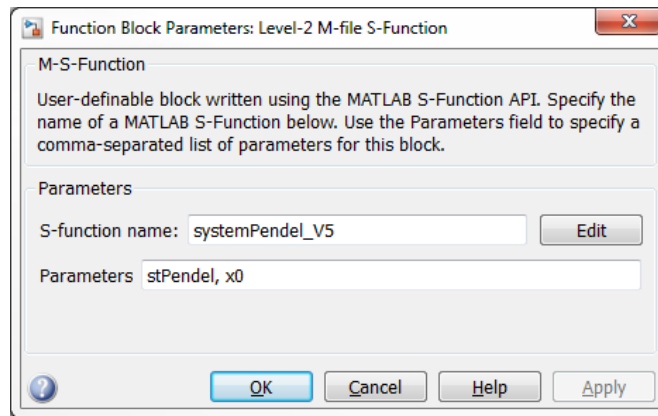


Abbildung 11.1: Verwendung der s-Function systemPendel_V5

- Variieren Sie bei der Simulation auch die Pendelparameter. (Berechnen Sie aber nicht eine neue Steuerung!) Welche Pendelparameter haben einen großen Einfluss auf das Ergebnis?

11.3 Protokoll

Das Protokoll soll die unter den Punkten „Auswertung“ angesprochenen Themen behandeln und die gestellten Fragen beantworten.

Außerdem fügen Sie bitte einen Screenshot des Simulink-Modells sowie (unkommentiert) den Code Ihrer Funktionen dem Protokoll bei.

12 Programmierung

12.1 Berechnen von $u^*(t)$ und $x^*(t)$

Listing 12.1: berechneTrajektorie

```
function stTraj = berechneTrajektorie(stTraj, T)
    % Gibt eine Struktur mit Inhalt Uebergangszeit T, Zeitvektor t,
    % Zustandsvektor x und Steuerfolge u zurück

    % Initialisierung von Randwertproblem. Die persistent Variablen →
    % ←werden gesetzt
    RandwertproblemDGL(0, 0, 0, stTraj, T);

    % Erstellung von solinit, benötigt von bvp4c, siehe help bvp4c
    % x:           Zeitschritte von Null bis T
    % y:           linearer Verlauf für die Zustände vom Anfangszustand →
    % ←zum Endzustand
    %           Randwerte, hier Winkel von Null auf Pi, →
    % ←Winkelgeschwindigkeit von Null auf Null
    % parameters: Annahme irgendwelcher Werte für die zwei freien →
    % ←Parameter p1 und p2 im Steuergesetz
    solinit.x = linspace( 0, T, 1000 );
    solinit.y = [(linspace( 0, pi, length(solinit.x))); zeros(1, length(→
    % ←solinit.x))];
    solinit.parameters = [0, 0];

    % sol:         Berechnung des Randwertproblems. Rückgabewerte sind →
    % ←Verlauf
    %           der Zeit und der zugehörigen Zuständen und die →
    % ←berechneten freien
    %           Parameter, die zur gewünschten Lösung des
    %           Randwertproblems führen
    % bvp4c:       Es bekommt drei Argumente übergeben. Die drei müssen →
    % ←die
    %           gleiche Reinfolgen in ihren Eingangsargumenten →
    % ←besitzen.
    %           Somit muss auch in Randwerte ein Parameter für die →
    % ←freien
    %           Parameter dastehen. Hier: Zeit, Zustände, freie →
    % ←Parameter
    sol = bvp4c(@RandwertproblemDGL, @RandwertproblemRB, solinit);
```

```

% Zeitvektor der Lösung
t = sol.x;

33 % Zustände 3 und 4 (des Gesamtsystems!) können direkt aus Lösung
    % RWP entnommen werden.
    x3 = sol.y(1, :);
    x4 = sol.y(2, :);

38 % Auslesen der berechneten freien Parameter für Randwertproblem
    p1 = sol.parameters(1);
    p2 = sol.parameters(2);

43 % Das Steuerfolge wird für Zeitverlauf t berechnet
    u = ( 6*(-p1 - 3*p2).*(t./T) + ...
          12*(3*p1 + 8*p2).*(t./T).^2 + ...
          20*(-3*p1 - 6*p2).*(t./T).^3 + ...
          30*p1.*(t./T).^4 + 42*p2.*(t./T).^5 ) / (T^2);

48 % Zustände 1 und 2 werden für Zeitverlauf t berechnet
    x1 = (-p1 - 3*p2).*(t./T).^3 + ...
          (3*p1 + 8*p2).*(t./T).^4 + ...
          (-3*p1 - 6*p2).*(t./T).^5 + ...
          p1.*(t./T).^6 + p2.*(t./T).^7;

53
    x2 = ( 3*(-p1 - 3*p2).*(t./T).^2 + ...
          4*(3*p1 + 8*p2).*(t./T).^3 + ...
          5*(-3*p1 - 6*p2).*(t./T).^4 + ...
          6*p1.*(t./T).^5 + 7*p2.*(t./T).^6 ) / T;

58 % Rückgabestruktur mit allen benötigten Werten
    stTraj.T = T;
    stTraj.vT = t;
    stTraj.mX = [x1; x2; x3; x4];
    stTraj.vU = u;
    stTraj.P = sol.parameters;

end % function berechneTrajektorie

```

Listing 12.2: RandwertproblemDGL

```

function dxdt = RandwertproblemDGL(t, x, P, stPendel, T)
% t:      Zeit
% x:      Zustände x(1) und x(2)
% P:      freie Parameter p1 und p2
% Pendeldata: Pendeldata
% T:      Übergangszeit

8 persistent m s J g sT p1 p2;

```



```

13     if (nargin > 3)
        m = stPendel.mPendel;
        l = stPendel.lPendel;
        s = l/2;
        g = stPendel.g;
        J = 1/12 * stPendel.mPendel * stPendel.lPendel^2;
        sT = T;
        dxdt = 0;
18         return;
    end

    p1 = P(1);
    p2 = P(2);
23
    % Steuergesetz mit noch unbekannten Parametern p1 und p2
    u = ( 6*(-p1 - 3*p2)*(t/sT) + 12*(3*p1 + 8*p2)*(t/sT)^2 + ...
        20*(-3*p1 - 6*p2)*(t/sT)^3 + ...
        30*p1*(t/sT)^4 + 42*p2*(t/sT)^5 ) / (sT^2);
28
    % Differentialgleichung 2ter Ordnung, x(1): Winkel,
    % x(2): Winkelgeschwindigkeit
    dxdt = [ x(2) ; ...
        -m*s*g/(m*s^2+J)*sin(x(1)) ...
33         - m*s/(m*s^2+J)*cos(x(1))*u;    ]; % f(4) + g(4)→
        ← * u

end % function RandwertproblemDGL

```

Listing 12.3: RandwertproblemRB

```

function res = RandwertproblemRB(xa, xb, P) %#ok<INUSD>

    % Spaltenvektor mit den Randwerten
    % Zuerst die Anfangswerte, dann die Endwerte
5    res = [ xa(1); ...
        xa(2); ...
        xb(1)-pi; ...
        xb(2)];

10 end % function RandwertproblemRB

```

Listing 12.4: A_Berechnung

```

clear all;
clc;

%% Berechnung des Randwertproblems
5 stPendel = ladePendel();

```

```

% Übergangszeiten
%vT = 1.0:0.2:2.2; % (für Versuch 6)
10 vT = 1.18;
    %   Gute Lösungen für T = 1 bis 2 Sekunden,
    %   Ausprobieren!
    %   Erklärung Übergangszeit: T = 1.3 Sekunden soll das Pendel
    %   von der unteren Lage in die obere instabile Lage →
    %   ←gebracht
15    %   werden.

% Leeres Cell-Array passender Länge erzeugen
caTraj = cell(length(vT), 1);

20 % Trajektorien berechnen
for i = 1:length(vT)
    caTraj{i} = berechneTrajektorie( stPendel, vT(i) );
end
stTraj = caTraj{1};
25
%% Daten speichern
% (für Versuch 6)
%save( 'Trajektorien', 'caTraj' )

```

Listing 12.5: B_Ausgabe

```

% Darstellung der Lösung des berechneten Randwertproblems
2
T = stTraj.T;
vTGrenze = [0; T];
vX1Grenze = [1; 1] * 0.5;
vX2Grenze = [1; 1] * 1.6;
7 vUGrenze = [1; 1] * 12;
vPiGrenze = [1; 1] * pi;

% Ausgabe der Stellgrößenfolge
figure
12 subplot (5, 1, 1);
plot(stTraj.vT, stTraj.vU, ...
    vTGrenze, vUGrenze, 'k', vTGrenze, -vUGrenze, 'k');
title('Stellgrößenverlauf');
ylabel('u^{*}(t)_{[m/s^2]}');
17
% Ausgabe der Trajektorie Weg
subplot (5, 1, 2);
plot(stTraj.vT, stTraj.mX(1,:), ...
    vTGrenze, vX1Grenze, 'k', vTGrenze, -vX1Grenze, 'k');
22 title('Trajektorie_Weg');
ylabel('x_1^{*}(t)_{[m]}');

```

```

% Ausgabe der Trajektorie Geschwindigkeit
subplot (5, 1, 3);
27 plot(stTraj.vT, stTraj.mX(2,:), ...
      vTGrenze, vX2Grenze, 'k', vTGrenze, -vX2Grenze, 'k');
title('Trajektorie_Geschwindigkeit');
ylabel('x_2^(t)_[m/s]');

32 % Ausgabe der Trajektorie Winkel
subplot (5, 1, 4);
plot(stTraj.vT, stTraj.mX(3,:), ...
      vTGrenze, vPiGrenze, 'k');
title('Trajektorie_Winkel');
37 ylabel('x_3^(t)_[rad]');

% Ausgabe der Trajektorie Winkelgeschwindigkeit
subplot (5, 1, 5);
plot(stTraj.vT, stTraj.mX(4,:));
42 title('Trajektorie_Winkelgeschwindigkeit');
ylabel('x_4^(t)_[rad/s]');
xlabel('Zeit_t_[s]');

47 %animierePendel(stTraj.vT, stTraj.mX', Pendelraten, [], 0.01, 0);

```

12.2 Simulation

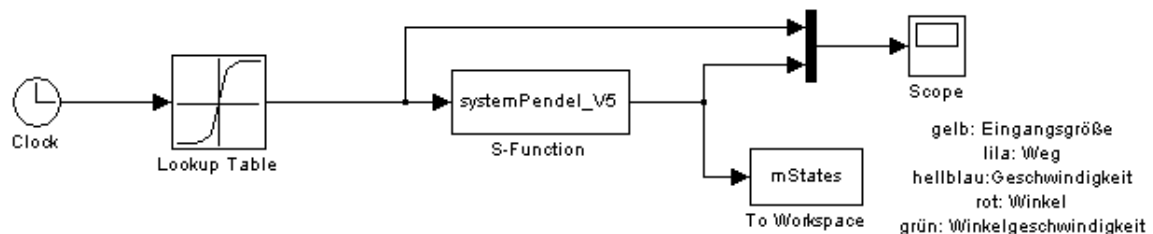


Abbildung 12.1: Steuerung unter Simulink

Listing 12.6: C_Simulation_mit_Simulink.m

```

% Parameter Pendel
stPendel = ladePendel();

3 stTraj = berechneTrajektorie(stPendel, 1.2);
% Verfälschen der Pendelraten
stPendel.lPendel = stPendel.lPendel;
%stPendel.mPendel = 10*stPendel.mPendel;
8 %stPendel.g = 10;

% Anfangszustand Pendel (0 ist unten, pi oben)
x0 = [0 0 0 0].';

```

```

13  t = stTraj.vT;
    u = stTraj.vU;
    T = stTraj.T;

    % Ausführen Simulation, Simulationszeit wird in vT gespeichert
18  vT = sim('SteuerungPendel');

    % =====
    % Verläufe plotten

23  vTGrenze = [0; T+1];
    vPiGrenze = [1; 1] * pi;

    % Ausgabe der Trajektorie Winkel
28  subplot (2, 1, 1);
    plot(stTraj.vT, stTraj.mX(3,:), ':', vT, mStates(:, 3), 'b', ...
         vTGrenze, vPiGrenze, 'k');
    title('Trajektorie_Winkel');
    ylabel('x_3^(t)/x_3(t)_[rad]');
33  legend('x_3^*', 'x_3' , 'Location', 'SouthEast')

    % Ausgabe der Trajektorie Winkelgeschwindigkeit
    subplot (2, 1, 2);
    plot(stTraj.vT, stTraj.mX(4,:), ':', vT, mStates(:, 4), 'b');
38  title('Trajektorie_Winkelgeschwindigkeit');
    ylabel('x_4^(t)/x_4(t)_[rad/s]');
    legend('x_4^*', 'x_4' , 'Location', 'NorthEast')
    xlabel('Zeit_t_[s]');

43  % =====
    % Animation abspielen
    % (Das 5. Argument ist die Zeit zwischen zwei Bildern in
    % Sekunden, das letzte ein "Schalter" zum Speichern der
48  % Einzelbilder.)
    % mStates wird mit Simulink berechnet

    animierePendel(vT, mStates, stPendel, [], 0.1, 0);

```

Zusatz: Simulation ohne Simulink

Listing 12.7: D_Simulation_ohne_Simulink.m

```

1  % Berechnung der Differentialgleichung für die berechneten Parameter

    T = stTraj.T;

```

```

% Initialisierung der Differentialgleichung
6 SchlittenpendelGesteuertDGL(0, 0, ladePendel(), stTraj);

[t2 , f] = ode45(@SchlittenpendelGesteuertDGL, [0 T+1], [0, 0, 0, 0]);
% Lösen der Differentialgleichung für Zeit T+x und Anfangswerte, →
    ←verschiedene Solver, verschiedene
11 % Ergebnisse, wie was auch die Darstellung mit simPendel deutlich zeigt
% mit ode45 fällt das Pendel nach links, mit ode23 nach rechts wieder →
    ←runter

subplot(4,1,1)
plot(t2, f(:,1))
16 title('Trajektorie: Weg des Schlittens')
axis tight

subplot(4,1,2)
plot(t2, f(:,2))
21 title('Trajektorie: Geschwindigkeit des Schlittens')
axis tight

subplot(4,1,3)
plot(t2, f(:,3))
26 title('Trajektorie: Winkel des Pendels')
axis tight

subplot(4,1,4)
plot(t2, f(:,4))
31 title('Trajektorie: Winkelgeschwindigkeit des Pendels')
axis tight
% Plotten aller Zustände

animierePendel(t2, f, ladePendel(), [], 0.025, 0)
36 % Aufruf der graphischen Simulation

% Aufräumen des Workspace
clear T t2 f;

```

Listing 12.8: SchlittenpendelGesteuertDGL

```

function dxdt = SchlittenpendelGesteuertDGL(t, x, stPendel, stTraj)    %→
    ← t zeit, x Zustände

    persistent m s g J p1 p2 T

5    if (nargin > 3)

        M = stPendel.mSchlitten;
        m = stPendel.mPendel;

```

```

10      l = stPendel.lPendel;
      s = l/2;
      g = stPendel.g;
      J = 1/12 * stPendel.mPendel * stPendel.lPendel^2;

      T = stTraj.T;
15      p1 = stTraj.P(1);
      p2 = stTraj.P(2);
      dxdt = 0;
      return

20  end

u = ( 6*(-p1 - 3*p2)*(t/T) + 12*(3*p1 + 8*p2)*(t/T)^2 + 20*(-3*p1 - 6*→
    ←p2)*(t/T)^3 +...
    30*p1*(t/T)^4 + 42*p2*(t/T)^5 ) / (T^2);

25  % u nur Zeitintervall der Eingangssteuerfolge berücksichtigen,
    % danach u = 0
    if (t <= T )
        dxdt = [
            x(2);
30            u ;
            x(4) ;
            -m*s*g/(m*s^2+J)*sin(x(3)) - m*s/(m*s^2+J)*cos(x(3))*u; % f→
                ←(4) + g(4) * u
            ];
    else
35        dxdt = [
            x(2);
            0 ;
            x(4) ;
            -m*s*g/(m*s^2+J)*sin(x(3)) ; % f(4)
40            ];
    end

end % function SchlittenpendelGesteuertDGL

```

13 Auswertung

13.1 Berechnen von $u^*(t)$ und $\mathbf{x}^*(t)$

Die Abbildungen 13.1, 13.2 und 13.3 zeigen die berechneten Verläufe für die Eingangsgröße und die Zustandstrajektorien bei den Übergangszeiten $T = 1,0\text{ s}$, $1,18\text{ s}$ und $2,0\text{ s}$. In die Kurven von $u^*(t) = \ddot{x}$, $x_1^*(t) = x$ und $x_2^*(t) = \dot{x}$ sind auch die einzuhaltenden Begrenzungen durch waagrechte Linien eingezeichnet. In den Graphen von $x_3^*(t) = \varphi$ markiert eine Linie den Wert π .

Durch Probieren lässt sich die „optimale“ Übergangszeit, die die gegebenen Begrenzungen einhält, mit etwa $T = 1,18\text{ s}$ herausfinden.

Im Bereich $1,0\text{ s} \leq T \leq 2,0\text{ s}$ findet der Löser `bvp4c` relativ sicher eine Lösung des Randwertproblems. Wo- bei es auch vorkommen kann, dass bei manchen Zeiten innerhalb dieser Grenzen keine Lösung gefunden wird.

Bspw. findet der Löser für $T = 1,1\text{ s}$ mit den gegebenen Anfangswerten keine Lösung und bricht mit der Warnung ab, dass die geforderten Toleranzen nicht eingehalten werden können. Dies ist auf die Numerik zurückzuführen, d. h. es ist davon auszugehen, dass eine Lösung existiert, diese nur nicht gefunden wird. Für $T = 1,0\text{ s}$ und auch $T = 1,09\text{ s}$ und $T = 1,1001\text{ s}$ wird jeweils eine Lösung gefunden. Schaut man sich die berechneten Verläufe für $T = 1,1\text{ s}$ (Abbildung 13.4) an, so sieht man, dass sich der Löser etwas „verrannt“ hat. (Das Pendel überschlägt sich mehrmals.)

Für große Übergangszeiten T kann aus physikalischen Gründen keine Lösung gefunden werden. Dies kann man darüber begründen, dass das Pendel ja nicht beliebig langsam in die obere Lage gebracht werden kann, da es auf dem Weg einfach wieder Herunterfallen würde, wenn keine Beschleunigungen mehr auf das Pendel einwirken.

Die Menge der möglichen Lösungen wird auch dadurch eingeschränkt, dass der Winkel in der Endlage π betragen muss, obwohl im Grunde auch ungerade Vielfache ($k \cdot \pi$ mit k ungerade) möglich wären. Solche Lösungen können mit dem verwendeten Löser aber nicht gefunden werden (bzw. müsste vorher genau ein Vielfaches vorgegeben werden). Dies gilt auch dann, wenn die Funktion mit den Randbedingungen `RandwertproblemRB` so angepasst würde, dass bspw. immer die Abweichung zum nächsten ungeraden Vielfachen für den Winkel zurückgeben würde. Das liegt daran, dass `bvp4c` nur minimale Abweichungen der Randbedingungen zulässt, und dann sofort „gegensteuert“, so dass eine Abweichung von $\pi/2$ nie erreicht wird.

13.2 Simulation

Abbildung 13.5 zeigt die Ergebnisse der Simulation bei der genau dasselbe Modell wie für die Berechnung der Steuerung verwendet wurde. Man erkennt, dass das Pendel der Solltrajektorie sehr gut folgt (die gepunktete Linie ist gar nicht zu erkennen) und die obere Lage zunächst erreicht, nach kurzer Zeit jedoch wieder herunterfällt. Dies lässt sich damit erklären, dass aufgrund der auftretenden numerischen Ungenauigkeiten eine (wenn auch nur sehr, sehr kleine) Abweichung der Lage bei $t = T$ von der exakten oberen Ruhelage auftritt. Da das System nicht geregelt ist, führt dies zum Herunterfallen des Pendels.

Variiert man die Pendelmasse des simulierten Modells, dann erhält man dasselbe Ergebnis wie beim Originalmodell. In der Simulation zu den Ergebnissen aus Abbildung 13.6 wurde die Pendelmasse für

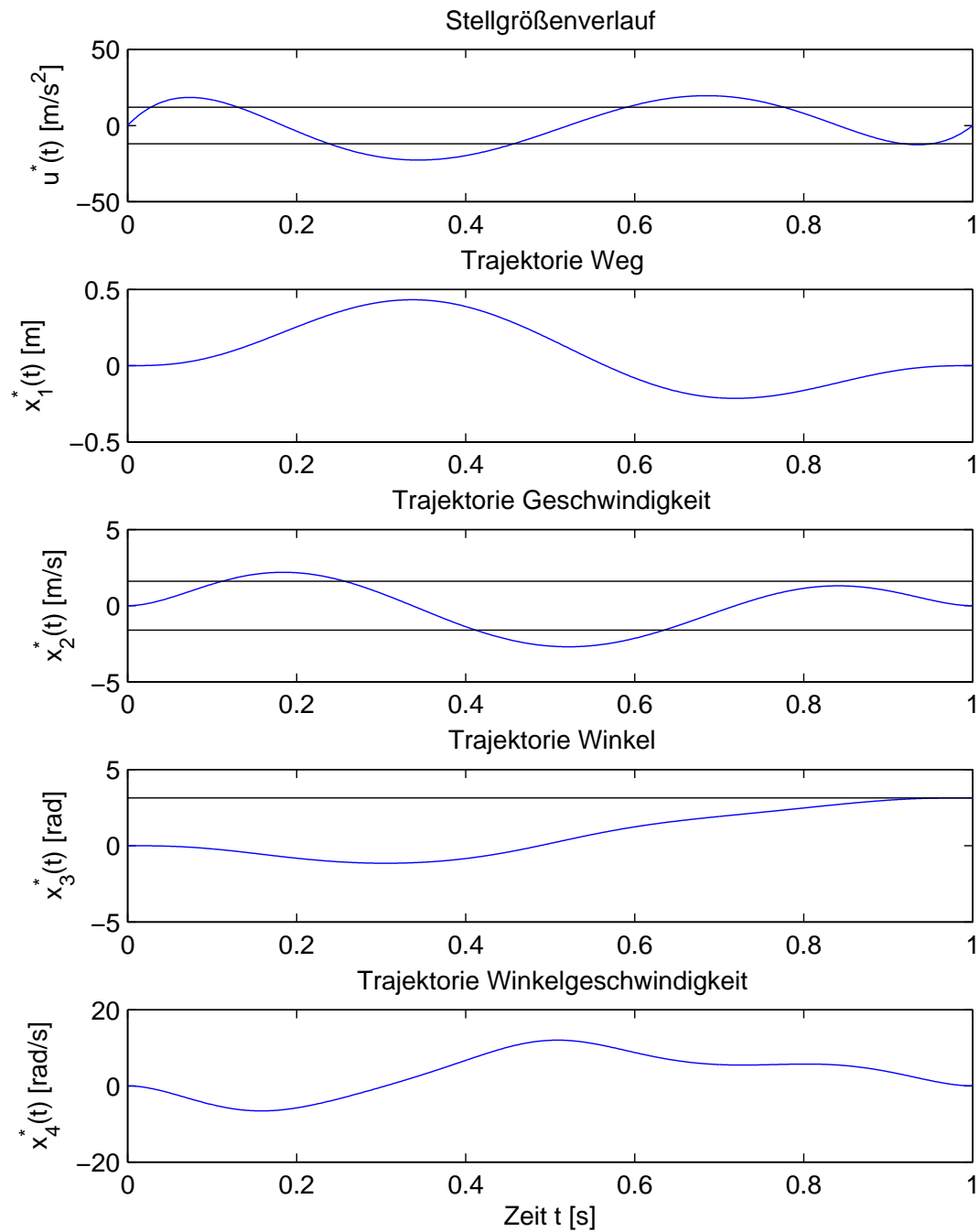


Abbildung 13.1: Berechnete Steuerfolge und Trajektorie für $T = 1,0\text{s}$

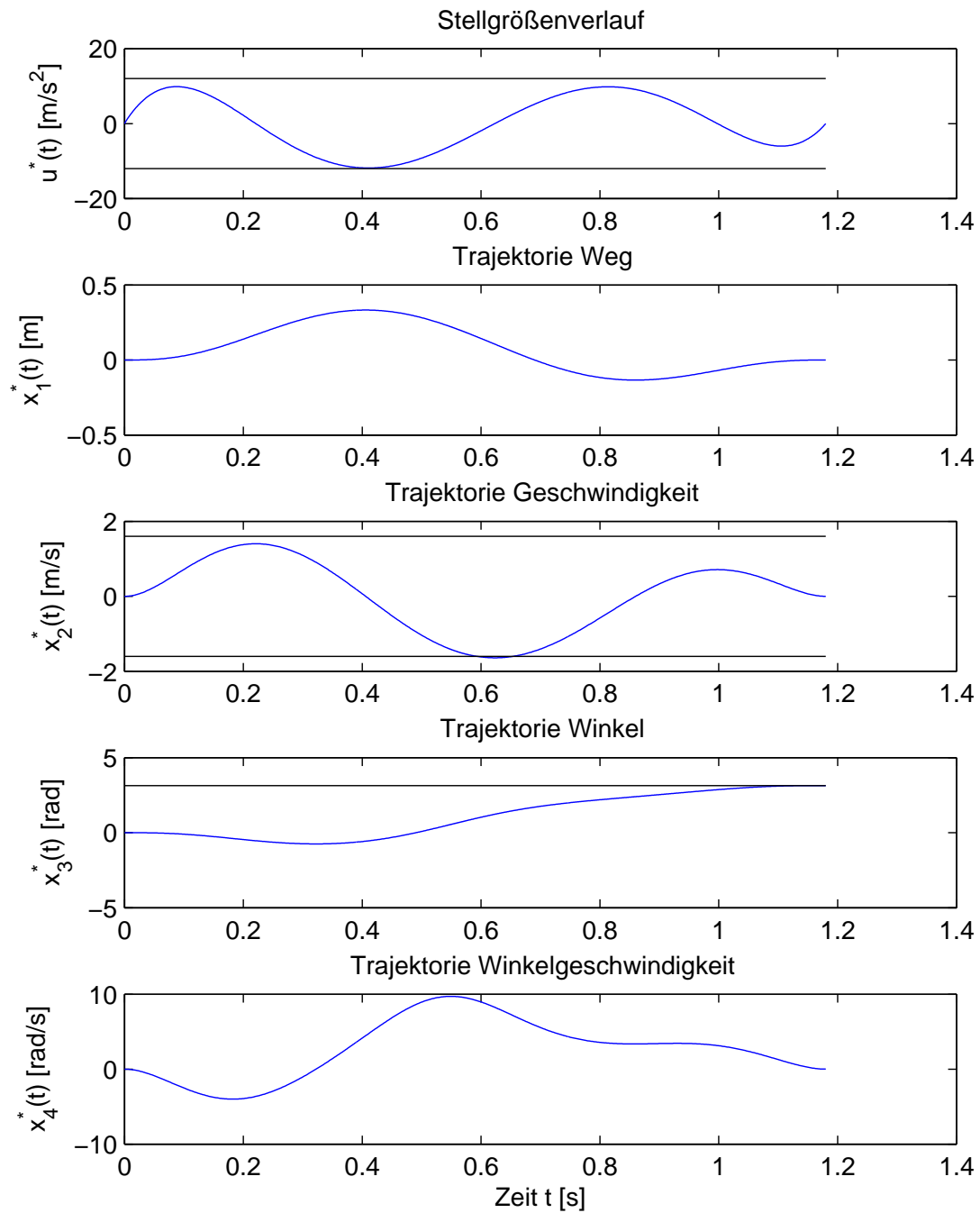


Abbildung 13.2: Berechnete Steuerfolge und Trajektorie für $T = 1,18 \text{ s}$

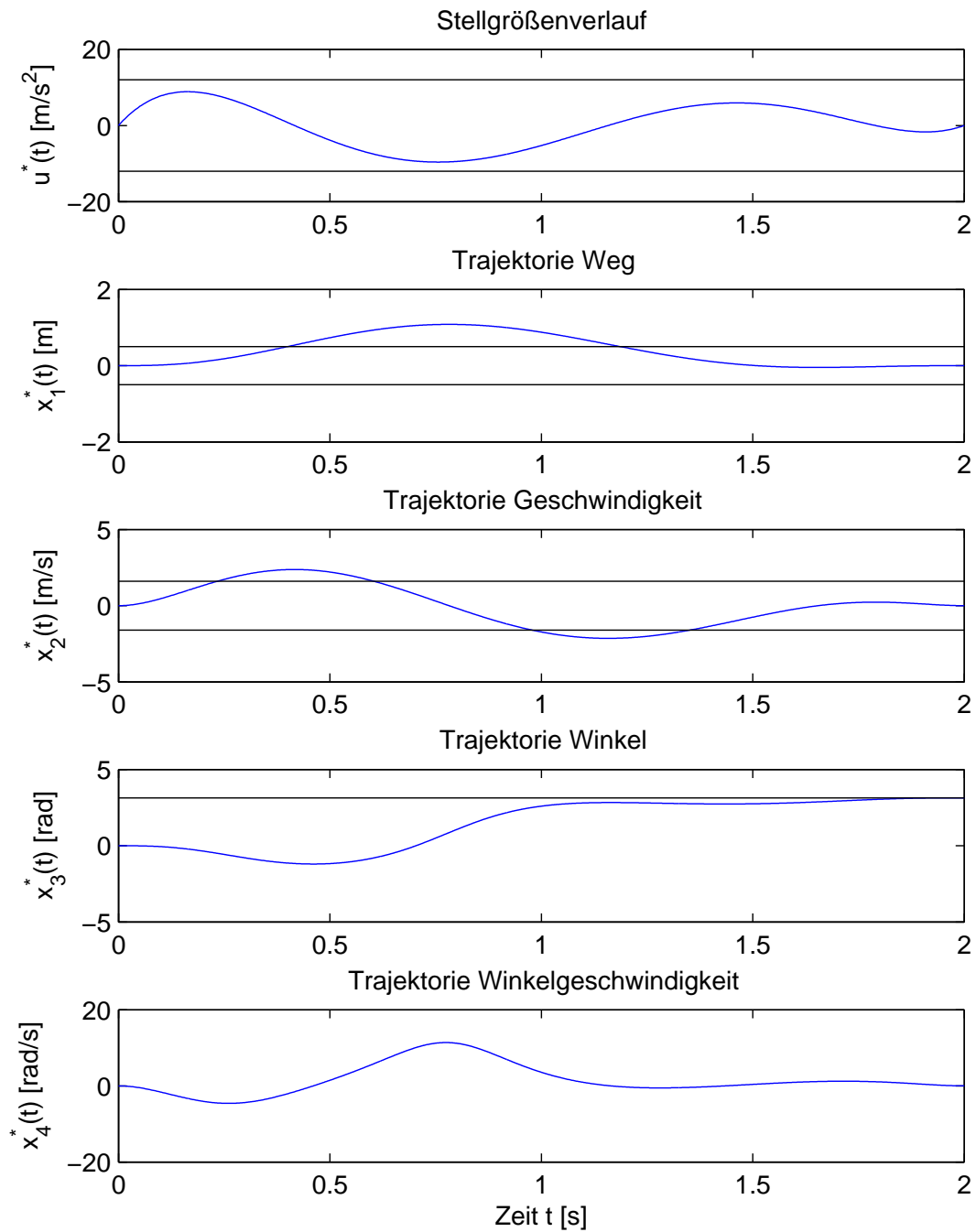


Abbildung 13.3: Berechnete Steuerfolge und Trajektorie für $T = 2,0$ s

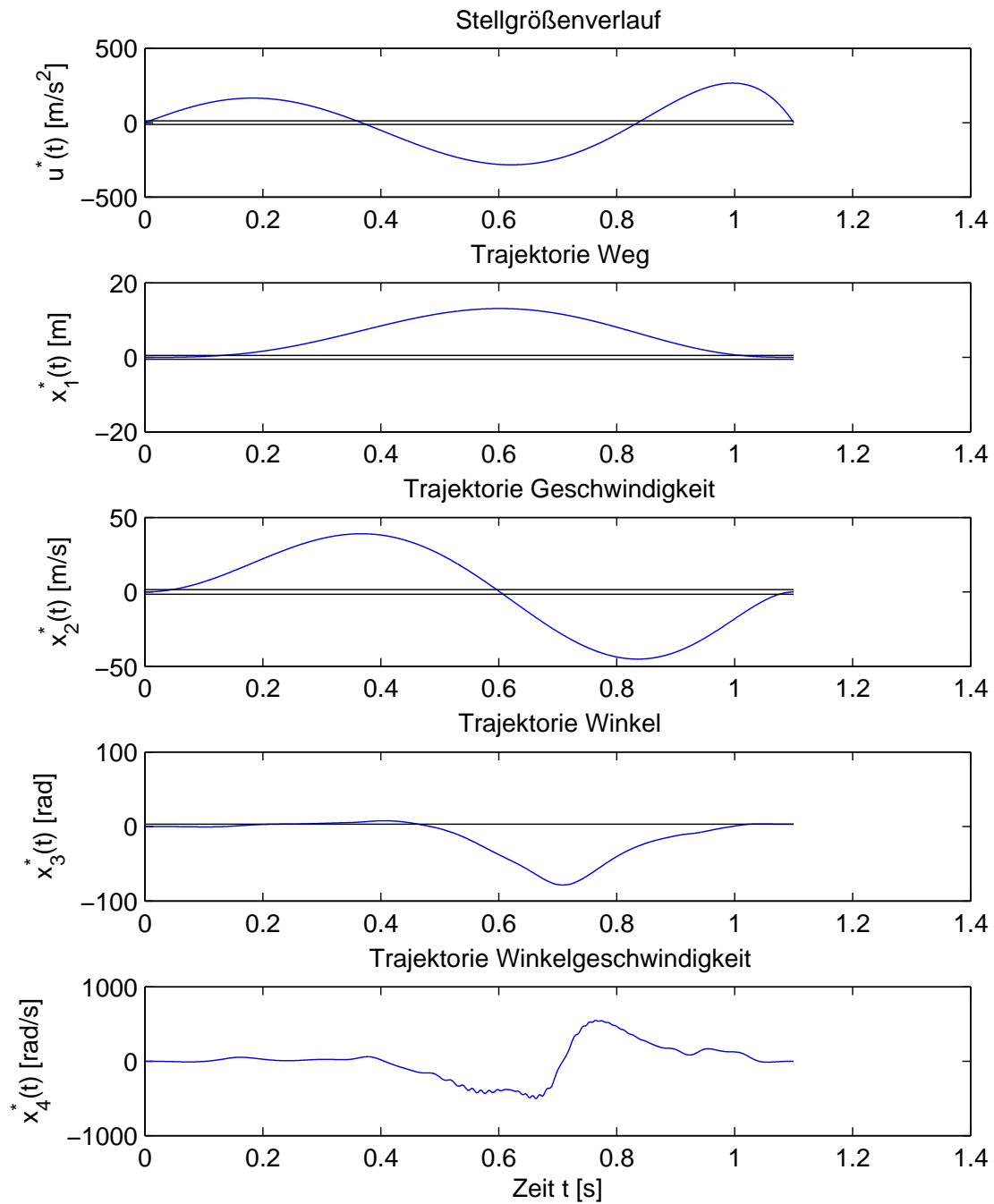


Abbildung 13.4: Steuerfolge und Trajektorie für $T = 1,1$ s nach Aufgabe des Löser

die Simulation verzeihnfacht. Dieses Verhalten kommt daher, dass in diesem Fall die Beschleunigung des Schlittens, und nicht die Kraft auf den Schlitten, als Eingang vorgegeben wird. Dadurch lautet die Zustandsraumdarstellung

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} x_2 \\ 0 \\ x_4 \\ -\frac{m_p \frac{l}{2} g \sin x_3}{m_p \frac{l^2}{4} + J} \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 0 \\ -\frac{m_p \frac{l}{2} \cos x_3}{m_p \frac{l^2}{4} + J} \end{bmatrix} u \quad (13.1)$$

$$y = x_1. \quad (13.2)$$

Wenn man sich noch bewusst macht, dass die Masse m_p linear in das Massenträgheitsmoment J eingeht (in diesem Fall ist $J = \frac{m_p l^2}{12}$), dann kann man die Masse in der letzten Gleichung von (13.1) kürzen. Damit kommt aber die Masse m_p gar nicht mehr in den Gleichungen vor, d.h. also, dass die Masse keinen Einfluss auf die Steuerung hat. (Wenn die Kraft auf den Schlitten vorgegeben würde, dann hätte die Masse natürlich einen Einfluss auf das Ergebnis.)

Die Länge des Pendels hingegen hat einen Einfluss auf das Systemverhalten, was man auch direkt aus (13.1) sehen kann. Es lässt sich nämlich die Länge l nicht aus den Brüchen kürzen. In Abbildung 13.7 sind die Ergebnisse zu sehen, wenn der Stab 5 % länger ist als zur Berechnung der Steuerung angenommen. Bei der Simulation zu Abbildung 13.8 hingegen ist der Stab 5 % kürzer. Durch den längeren Stab ist das Massenträgheitsmoment größer, und entsprechend reicht die Steuerung nicht aus, den Stab ganz in die senkrechte Position zu bringen, sondern dieser fällt wieder zurück. Umgekehrt führt das zu kleine Massenträgheitsmoment des Pendels im zweiten Fall dazu, dass es überschlägt.

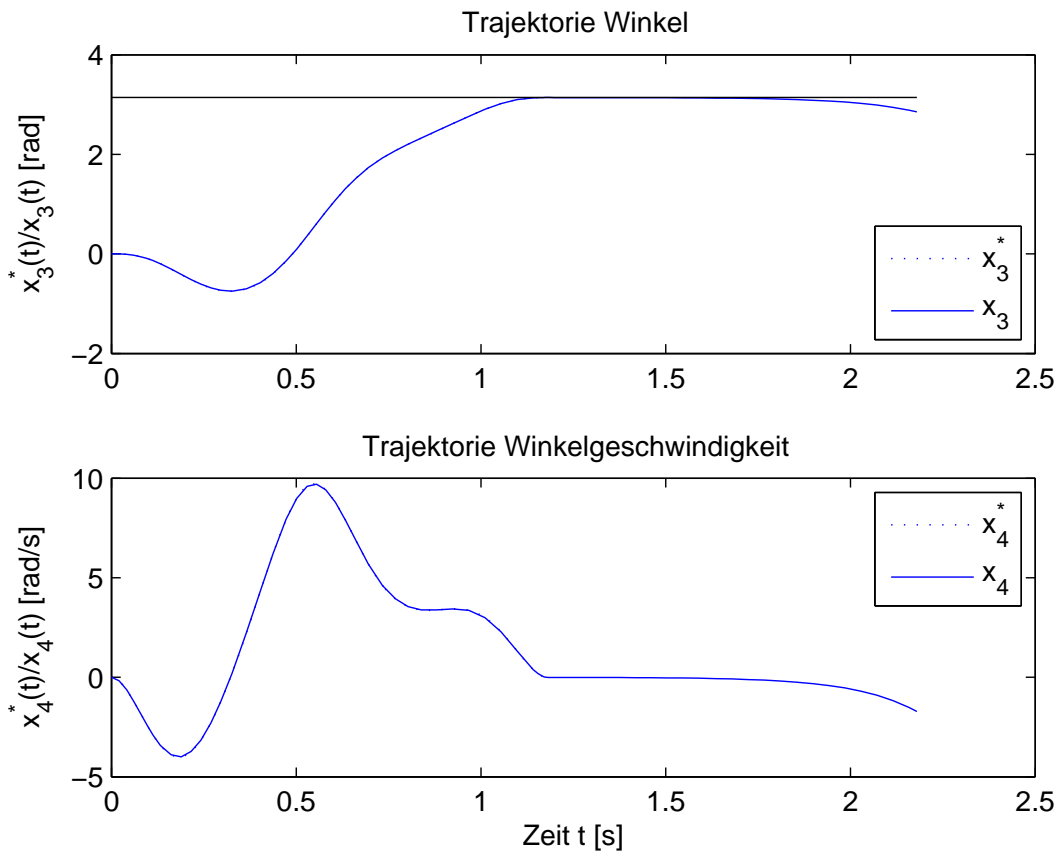


Abbildung 13.5: Simulation der Steuerung für $T = 1,18$ s, Modell wie für Berechnung der Steuerung

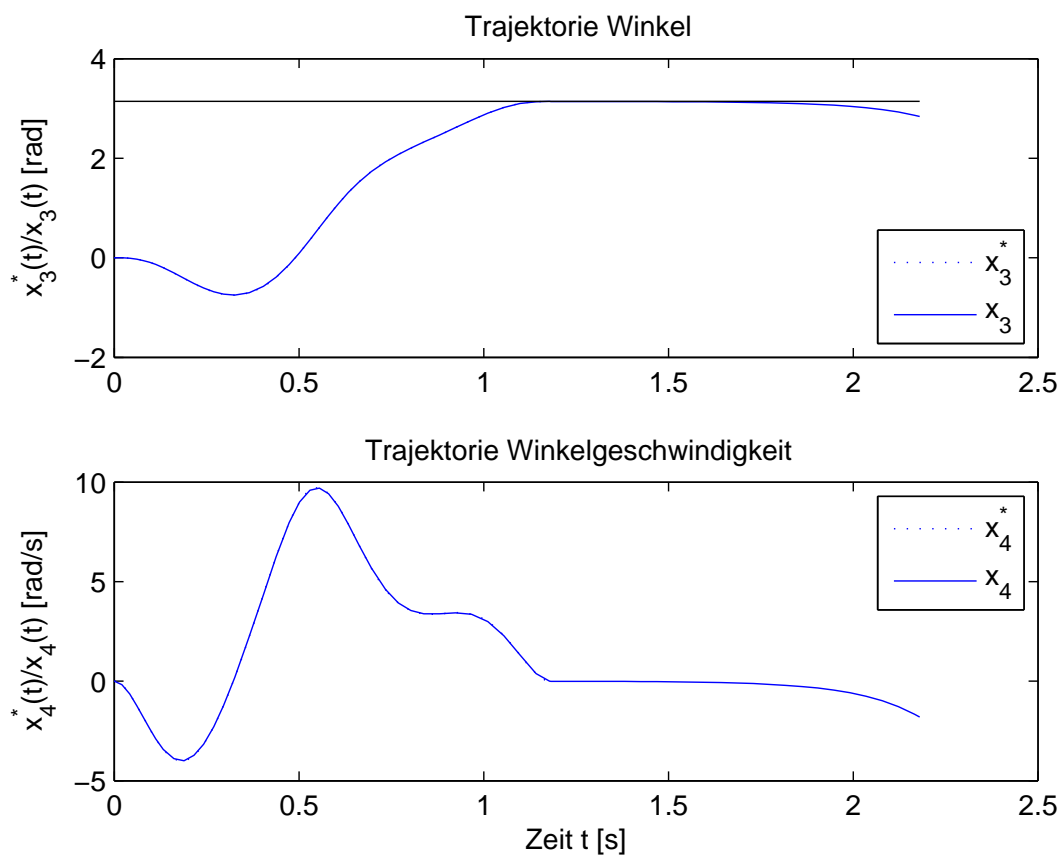


Abbildung 13.6: Simulation der Steuerung für $T = 1,18$ s, Modell mit zehnfacher Pendelmasse

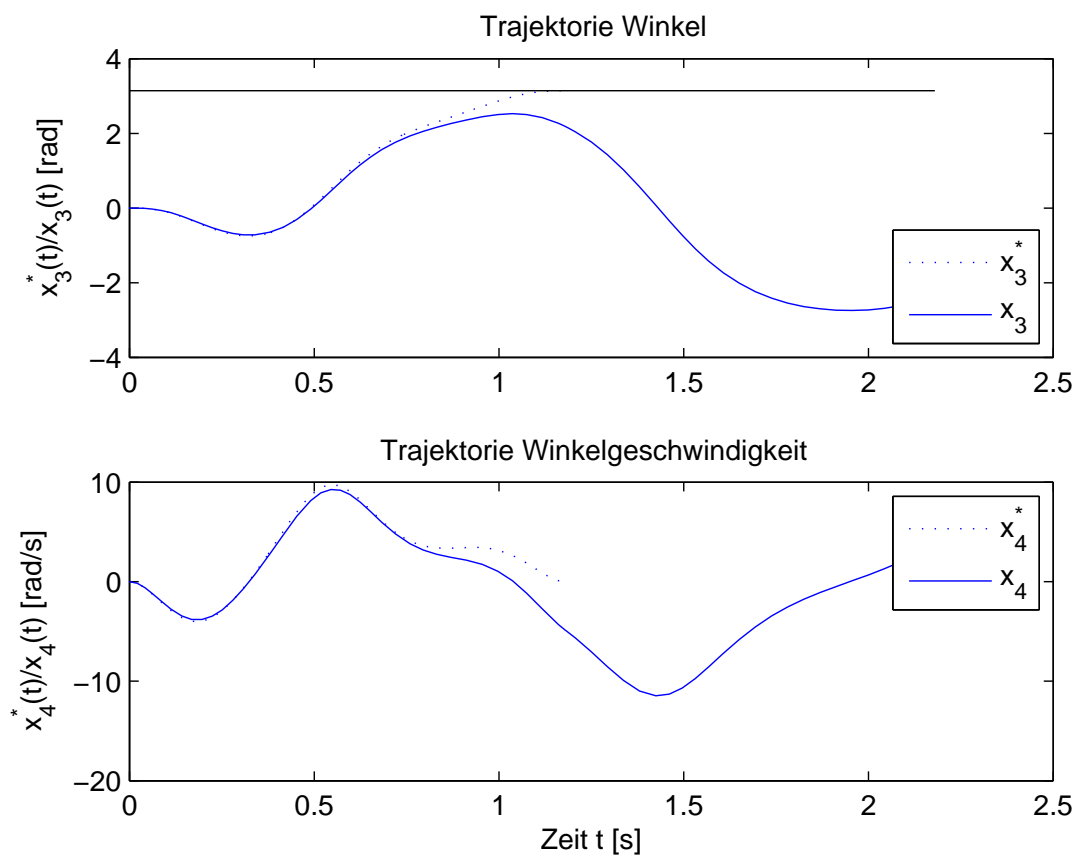


Abbildung 13.7: Simulation der Steuerung für $T = 1,18$ s, Modell mit 5 % längerem Stab

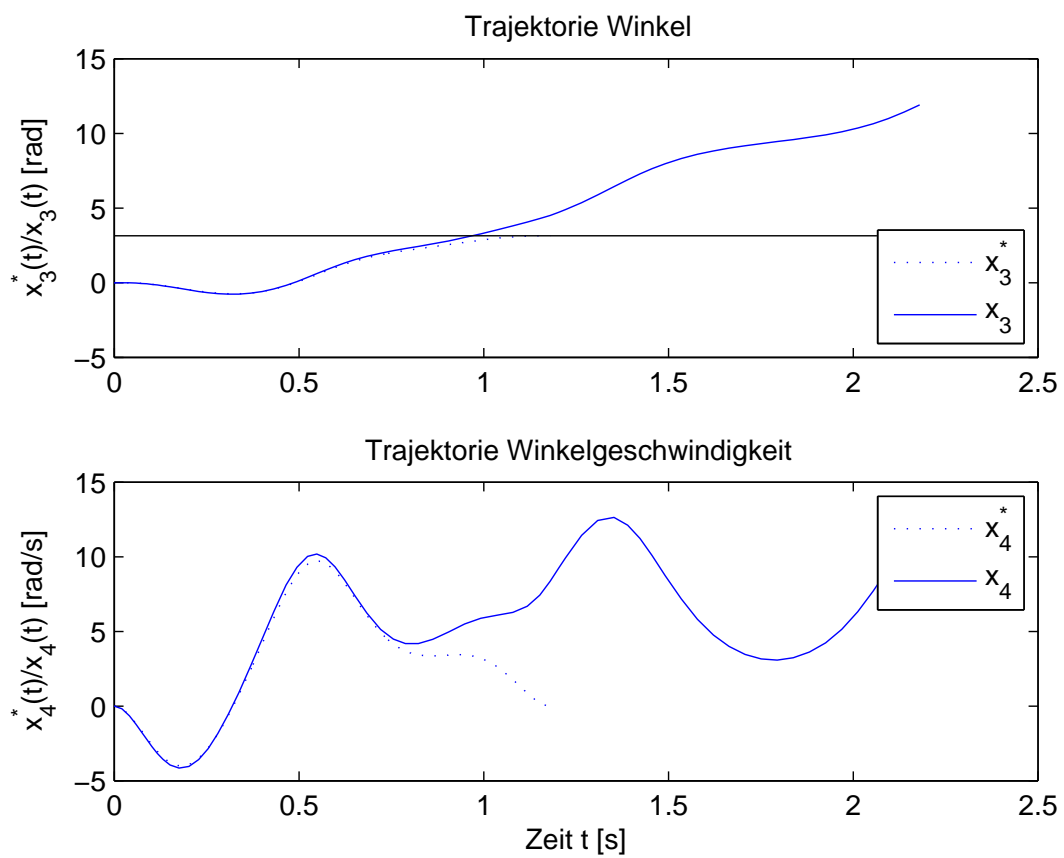


Abbildung 13.8: Simulation der Steuerung für $T = 1,18$ s, Modell mit 5 % kürzerem Stab

13.3 s-Function systemPendel_V5

Listing 13.1: Auszug aus systemPendel_V5

```
% *****
2 % Ableitungen berechnen
% *****
function Derivatives(block)

    % Parameter auslesen
7   pendelDaten = block.DialogPrm(1).Data;
    m = pendelDaten.mPendel;
    l = pendelDaten.lPendel;
    g = pendelDaten.g;

12  % Zustände auslesen
    x      = block.ContStates.Data;
    xs      = x(1);
    xs_d    = x(2);
    phi     = x(3);
17  phi_d   = x(4);

    % Eingang auslesen
    u = block.InputPort(1).Data(1);

22

    J = 1/12 * m * l^2;

    % Ableitungen berechnen
    x_d = [ xs_d;
27         u;
           phi_d;
           - m * l/2 * g * sin(phi) / (m*l^2/4 + J) - m * l/2 * cos(phi) /→
             ← ( m * l^2/4 + J ) * u;];

    % Ableitungen zuweisen
32  block.Derivatives.Data = x_d;

end
```

Versuch 6

Trajektorienfolgeregelung

14 Versuchsdurchführung	112
14.1 Linearisierung	112
14.2 Berechnen von $K(t)$	112
14.3 Folgeregelung unter Simulink aufbauen	113
14.4 Einfluss des Antriebs	113
14.5 Protokoll	114
15 Programmierung	115
15.1 Linearisierung	115
15.2 Simulation	118
15.3 Reglerverläufe für verschiedene Übergangszeiten	120
16 Auswertung	129
16.1 Beschleunigungsgesteuertes Modell	129
16.2 Kraftgesteuertes Modell	129

14 Versuchsdurchführung

14.1 Linearisierung

Es wird auch in diesem Versuch (zunächst) von den im Skript zum letzten Versuch gegebenen Systemgleichungen

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} x_2 \\ 0 \\ x_4 \\ -\frac{m_P \frac{l}{2} g \sin x_3}{m_P \frac{l^2}{4} + J} \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 0 \\ -\frac{m_P \frac{l}{2} \cos x_3}{m_P \frac{l^2}{4} + J} \end{bmatrix} u \quad (14.1)$$

$$y = x_1 \quad (14.2)$$

ausgegangen ($J = \frac{1}{12} m_P l^2$). D. h. die Eingangsgröße ist die Beschleunigung des Schlittens.

Bestimmen Sie analytisch die Matrizen

$$\mathbf{A}(\mathbf{x}, u) = \left. \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \right|_{\mathbf{x}, u} \quad (14.3)$$

$$\mathbf{B}(\mathbf{x}, u) = \left. \frac{\partial \mathbf{f}}{\partial \mathbf{u}} \right|_{\mathbf{x}, u}. \quad (14.4)$$

Schreiben Sie eine Funktion

$$[\mathbf{A}, \mathbf{B}] = \text{linPendelZR2}(\text{stPendel}, \mathbf{x}, u)$$

die die Matrizen $\mathbf{A}(\mathbf{x}, u)$ und $\mathbf{B}(\mathbf{x}, u)$ des um den gegebenen Arbeitspunkt \mathbf{x} und u linearisierten Systems zurückgibt.

14.2 Berechnen von $\mathbf{K}(t)$

Schreiben Sie eine Funktion

$$\mathbf{vPdot} = \text{RiccatiDGL}(t, \mathbf{vP}, \text{stPendel}, \text{stTraj}, Q, R)$$

die die RICCATIdifferentialgleichung (20.10) implementiert.

Diese Funktion muss vom DGL-Löser in der Form $\mathbf{vPdot} = \text{RiccatiDGL}(t, \mathbf{vP})$ aufrufbar sein. (Methoden dazu wurden in Versuch 5 vorgestellt.)

Dazu müssen in der Funktion `RiccatiDGL` folgende Schritte implementiert werden:

- Aus den in `stTraj` gespeicherten Daten der berechneten Trajektorie den zum Zeitpunkt t gehörenden Sollzustand und Eingangswert ermitteln (interpolieren).

- Die Systemmatrizen des linearisierten Modelles mit `linPendelZR2` berechnen.
- $\dot{\mathbf{P}}$ mit (20.10) berechnen. Dabei beachten, dass die Matrix \mathbf{P} in `vP` als Vektor übergeben wird und die Matrix $\dot{\mathbf{P}}$ als Vektor in `vPdot` zurückgegeben werden muss.

Erstellen Sie eine Funktion

$$[\mathbf{vTK}, \mathbf{mK}] = \text{berechneK}(\text{stPendel}, \text{stTraj}, Q, R)$$

die in `mK` den Reglervektor für die in `vTK` angegebenen Zeitpunkten zurückgibt.

Dazu müssen in dieser Funktion folgende Schritte implementiert werden:

- Berechnen des Anfangswertes $\mathbf{P}(T)$ durch Lösen von (20.11).
- Berechnen des Verlaufs von $\mathbf{P}(t)$ durch Lösen der RICCATI Differentialgleichung (20.10). (Hierzu wird die Funktion `RiccatiDGL` benötigt.)
- Berechnen des Verlaufes von $\mathbf{K}(t)$ mit (20.9).

14.3 Folgeregelung unter Simulink aufbauen

Erweitern Sie Ihr Simulink-Modell aus der vorherigen Übung um die Folgeregelung.

Es ist empfehlenswert, das Modell so aufzubauen, dass die Regelung schnell über eine Variable oder einen manuellen Schalter aktiviert und deaktiviert werden kann. So kann das unterschiedliche Verhalten schnell verglichen werden.

Simulieren Sie auch jetzt etwa eine Sekunde länger als die Übergangszeit T . Verändern Sie auch wieder die Pendelparameter des Simulationsmodells (ohne die Steuerung und Regelung neu zu berechnen).

Welche Unterschiede stellen Sie beim Vergleich der reinen Steuerung und der Steuerung mit Folgeregelung fest?

14.4 Einfluss des Antriebs

Bei der Berechnung der Trajektorien und zur Bestimmung der zeitvarianten Reglermatrix $\mathbf{K}(t)$ wurde zur Vereinfachung angenommen, dass der Systemeingang u des Schlitten-Pendel-Systems die Schlittenbeschleunigung \ddot{x} ist. Praktisch erfolgt die Steuerung/Regelung jedoch über einen Elektromotor, der eine Spannung U als Eingangsgröße besitzt und eine Kraft F auf den Schlitten ausübt.

Um diesen Motor möglichst einfach zu berücksichtigen, verwenden Sie in einem ersten Schritt wieder die s-Function `systemPendel_V3` aus Versuch 3 und 4, die als Eingangsgröße die Kraft F erwartet. Die Steuerung (die nicht neu hergeleitet/berechnet werden soll) gibt aber natürlich immer noch eine Beschleunigung u^* vor, und auch der Reglerausgang Δu ist eine Beschleunigung. Um also die durch Steuerung und Regler geforderte Beschleunigung $u(t) = u^*(t) + \Delta u(t)$ in eine Kraft umzurechnen, multiplizieren Sie diese mit der Gesamtmasse des Schlitten-Pendel-Systems:

$$F(t) = u(t) \cdot (m_s + m_p).$$

(Dabei sind die Massen zu benutzen, die auch zur Berechnung der Trajektorien benutzt wurden.)

Vergleichen Sie das neue Systemverhalten mit dem aus Abschnitt 14.3, wieder ohne und mit Trajektorienfolgeregelung.

Im letzten Versuch 5 hat sich gezeigt, dass die Pendelmasse $m = m_p$ keinen Einfluss auf die Steuerung hat, wenn der Systemeingang die Beschleunigung ist. Wie verhält es sich wenn die Kraft F der Systemeingang ist?

Ist die Umrechnung der Schlittenbeschleunigung u in die Kraft F , die auf den Schlitten aufgebracht werden muss, exakt, wenn davon ausgegangen wird, dass die Pendelparameter m_s und m_p genau bekannt sind?

Tatsächlich wird aber auch keine Kraft vorgegeben, sondern die Spannung U am Motor. Ein Elektromotor ist aber wiederum ein dynamisches System. Modellieren Sie dieses näherungsweise durch ein PT_1 -Glied mit der Zeitkonstanten T_M .

Probieren Sie für verschiedene Werte für T_M aus, wie sich das gesteuerte System ohne und mit Folgeregelung verhält.

14.5 Protokoll

Gehen Sie bitte auf die angesprochenen Punkte und Fragen ein. Verwenden Sie zur Erläuterung geeignete Plots.

Fügen Sie bitte Screenshots der beiden Simulink-Modelle (mit `systemPendel_V3` und `systemPendel_V5`) sowie (unkommentiert) Ihre in diesem Versuch erstellen Funktionen.

15 Programmierung

15.1 Linearisierung

Aus

$$f(\mathbf{x}, u) = \begin{bmatrix} x_2 \\ 0 \\ x_4 \\ -\frac{m_P \frac{l}{2} g \sin x_3}{m_P \frac{l^2}{4} + J} \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 0 \\ -\frac{m_P \frac{l}{2} \cos x_3}{m_P \frac{l^2}{4} + J} \end{bmatrix} u,$$

$J = \frac{1}{12} m_P l^2$, ergibt sich

$$A(\mathbf{x}, u) = \frac{\partial f}{\partial \mathbf{x}} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -\frac{m_P \frac{l}{2} g \cos x_3}{m_P \frac{l^2}{4} + J} + \frac{m_P \frac{l}{2} \sin x_3}{m_P \frac{l^2}{4} + J} u & 0 \end{bmatrix}$$

und

$$B(\mathbf{x}, u) = \frac{\partial f}{\partial u} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ -\frac{m_P \frac{l}{2} \cos x_3}{m_P \frac{l^2}{4} + J} \end{bmatrix}.$$

Listing 15.1: linPendelZR2

```
1 function [A, B] = linPendelZR2(stPendel, vX0, u0)

    l = stPendel.lPendel;
    g = stPendel.g;

6     A = [0, 1, 0, 0;
           0, 0, 0, 0;
           0, 0, 0, 1;
           0, 0, -3/2/l*g*cos(vX0(3))+3/2/l*sin(vX0(3))*u0, 0];

11    B = [0;
          1;
          0;
          -3/2/l*cos(vX0(3))];

16 end
```

15.1.1 Berechnung $K(t)$

Listing 15.2: RiccatiDGL

```
function vPdot = RiccatiDGL(t, vP, stPendel, stTraj, Q, R)

    persistent s_stPendel s_stTraj s_Q s_Rinv;

4     if (nargin > 2)
        s_stPendel = stPendel;
        s_stTraj = stTraj;
        s_Q = Q;
9        s_Rinv = inv(R);

        vPdot = [];

        return;
14    end

    % Aus Zeit Arbeitspunkt berechnen
    vX0 = interp1(s_stTraj.vT, s_stTraj.mX.', t);
    vX0 = vX0.';

19    u0 = interp1(s_stTraj.vT, s_stTraj.vU.', t);

    % Um Arbeitspunkt linearisieren
    [A, B] = linPendelZR2(s_stPendel, vX0, u0);

24    n = length(A);

    P = reshape(vP, n, n);

29    Pdot = P * B * s_Rinv * B.' * P - P * A - A.' * P - s_Q;

    vPdot = reshape(Pdot, n*n, 1);

end
```

Listing 15.3: berechneK

```
function [vTK, mK] = berechneK(stPendel, stTraj, Q, R)

2    % Anfangswert (bzw. Endwert) von P aus algebraischer
    % Riccatigleichung bestimmen.

    vXe = [0, 0, pi, 0].';
7    u0 = 0;
    [Ae, Be] = linPendelZR2(stPendel, vXe, u0);

    S = zeros(size(Be));
```

```

12     E = eye(size(Ae));

13     Pe = care(Ae, Be, Q, R, S, E);

14     % Zeitpunkte festlegen, für die P(t) bestimmt werden soll
17     T = stTraj.T;
    % vTP = linspace(T, 0, 100);
    vTP = linspace(T, 0, length(stTraj.vT));

22     % Riccati-Differentialgleichung initialisieren
    RiccatiDGL(0, 0, stPendel, stTraj, Q, R);
    % Pe = reshape(Pe, size(Ae,1)*size(Ae,1),1)
    % Riccati-Differentialgleichung lösen
    [vTP, mP] = ode45(@RiccatiDGL, vTP, Pe);

27

    % Riccati-DGL wurde rückwärts gelöst, daher wieder
    % richtig rum drehen
    vTP = flipud(vTP);
    mP = flipud(mP);
32
    % figure
    % plot(vTP,mP)
    % K für alle Zeitpunkte bestimmen
    [vTK, mK] = getK(vTP, mP, stPendel, stTraj, R);
37 % figure
    % plot(vTK,mK)
end % function berechneK

42 % subfunction getK
function [vT, mK] = getK(vT, mP, stPendel, stTraj, R)

    n = sqrt(size(mP,2));

47     mK = zeros(length(vT), n);

    for i_t = 1:length(vT)
        t = vT(i_t);

52         % Aus Zeit Arbeitspunkt berechnen
        vX0 = interp1(stTraj.vT, stTraj.mX.', t);
        vX0 = vX0';

        u0 = interp1(stTraj.vT, stTraj.vU.', t);

57         % Um Arbeitspunkt linearisieren
        [~, B] = linPendelZR2(stPendel, vX0, u0);

```

Variable	Erläuterung
iForceInput	0: Beschleunigungsgesteuert, 1: Kraftgesteuert
iFeedbackOn	0: Regelung aus, 1: Regelung ein
Mges	Gesamtmasse Schlitten-Pendel-System
T_Motor	Zeitkonstante des Motors
stTraj	Trajektorie aus Versuch 5
vTK, mK	Reglervektoren $K(t)$ zu angegebenen Zeitpunkten
stPendel	Pendeldaten
x0	Anfangszustand ($= [0 \ 0 \ 0 \ 0]^T$)

Tabelle 15.1: Variablen für Simulink-Modell

```

62      % P wieder in Matrixform bringen
      P = reshape(mP(i_t, :), n, n);

      % K zum aktuellen Zeitschritt berechnen
      mK(i_t, :) = R \ B.' * P;
67  end
end % subfunction getK

```

15.2 Simulation

In der Musterlösung sind beide Varianten (Beschleunigungs- und Kraftgesteuert) in einem Modell erstellt. Es werden immer beide Modelle berechnet, und über den Switch wird entschieden, welcher Modellausgang (d. h. Zustand) zurückgeführt wird. Dadurch steigt zwar die Rechenzeit an, aber das ist hier unkritisch. Dafür ist es einfacher zu warten und zu bedienen.

In Tabelle 15.1 sind alle Variablen aufgelistet, die zur Simulation im Base-Workspace vorhanden sein müssen.

Im Block „MATLAB Fcn“ wird als Funktion

```
interp1(stTraj.vT, stTraj.mX.', u, 'nearest', 'extrap').'
```

und als Ausgangsdimension 4 eingetragen. Im Block „MATLAB Fcn1“ entsprechend

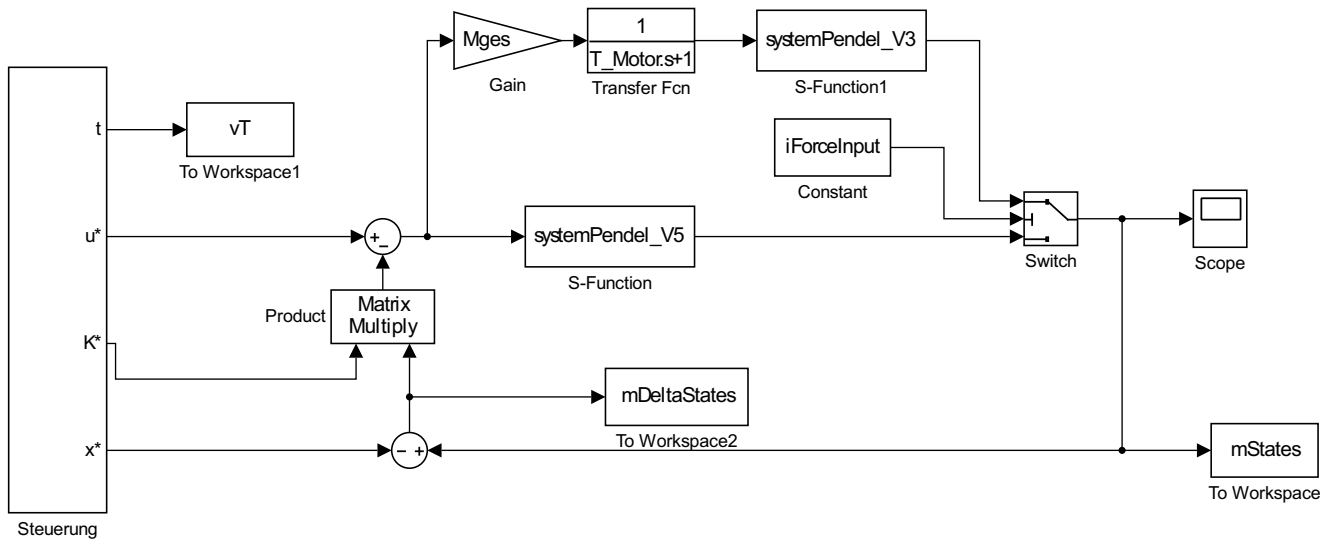
```
interp1(vTK, mK, u, 'nearest', 'extrap')
```

und ebenfalls 4. Durch die Wahl der Interpolationsmethode 'nearest' kann einfach extrapoliert werden, wenn die Simulationszeit t größer als die Übergangszeit T wird. Würde man 'linear' nehmen, dann würden die Werte „weglaufen“.

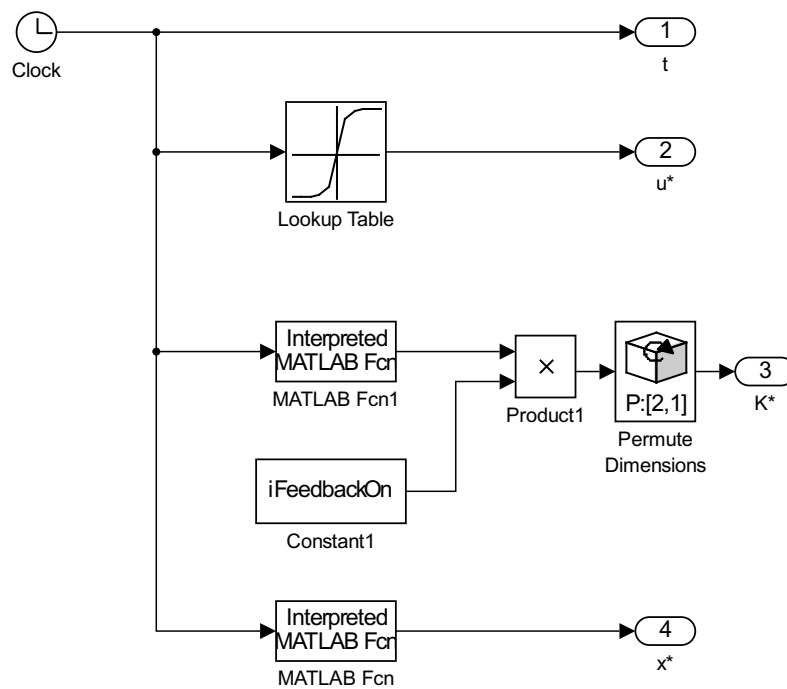
Listing 15.4 zeigt das Skript zum Ausführen des Modells. Es ist darauf zu achten, dass die Berechnung der Gesamtmasse vor dem Verfälschen der Pendeldaten erfolgt. (Die verwendete Trajektorie ist als stTraj in der mat-Datei „Traj.mat“ gespeichert.)

Listing 15.4: Versuch6

```
clear all
```

(a) Hauptmodul



(b) Submodel Steuerung

Abbildung 15.1: Steuerung mit Trajektorienfolgeregung unter Simulink

```

2  clc;

    % Trajektorie laden
    l = load( 'Trajektorien' );
    stTraj = l.caTraj{2};

7

    % Pendel laden
    stPendel = ladePendel();
    % Gesamtmasse unbedingt vor dem Verfälschen der
12 % Pendeldata berechnen.
    Mges = stPendel.mPendel + stPendel.mSchlitten;

    % Pendeldata verfälschen
    stPendel.lPendel = 1.05 * stPendel.lPendel;
17 stPendel.mPendel = 1 * stPendel.mPendel;

    % Trajektorienfolgeregelung benutzen? 0 or 1
    iFeedbackOn = 1;

22 % Modell mit Krafteingang benutzen? 0 or 1
    iForceInput = 0;
    % Motorkonstante
    T_Motor = 0.01;

27

    % Trajektorie laden
    %load Traj.mat

    Q = diag([1 1 1 1]);
32 R = 1;

    [vTK, mK] = berechneK(stPendel, stTraj, Q, R);

    % Aufschwung startet unten
37 x0 = [0 0 0 0].';

    sim('RegelungV6.slx', stTraj.T+6);

    animierePendel(vT, mStates, stPendel, [], 0.01, 0);

```

15.3 Reglerverläufe für verschiedene Übergangszeiten

Zum Vergleich während des Praktikums sind nachfolgend die Verläufe der Reglereinträge für verschiedene Übergangszeiten dargestellt. Für den Reglerentwurf wurden folgende Parameter gewählt

$$\mathbf{Q} = \text{diag}([100 \quad 1 \quad 100 \quad 1])$$
$$R = 1 \text{ .}$$

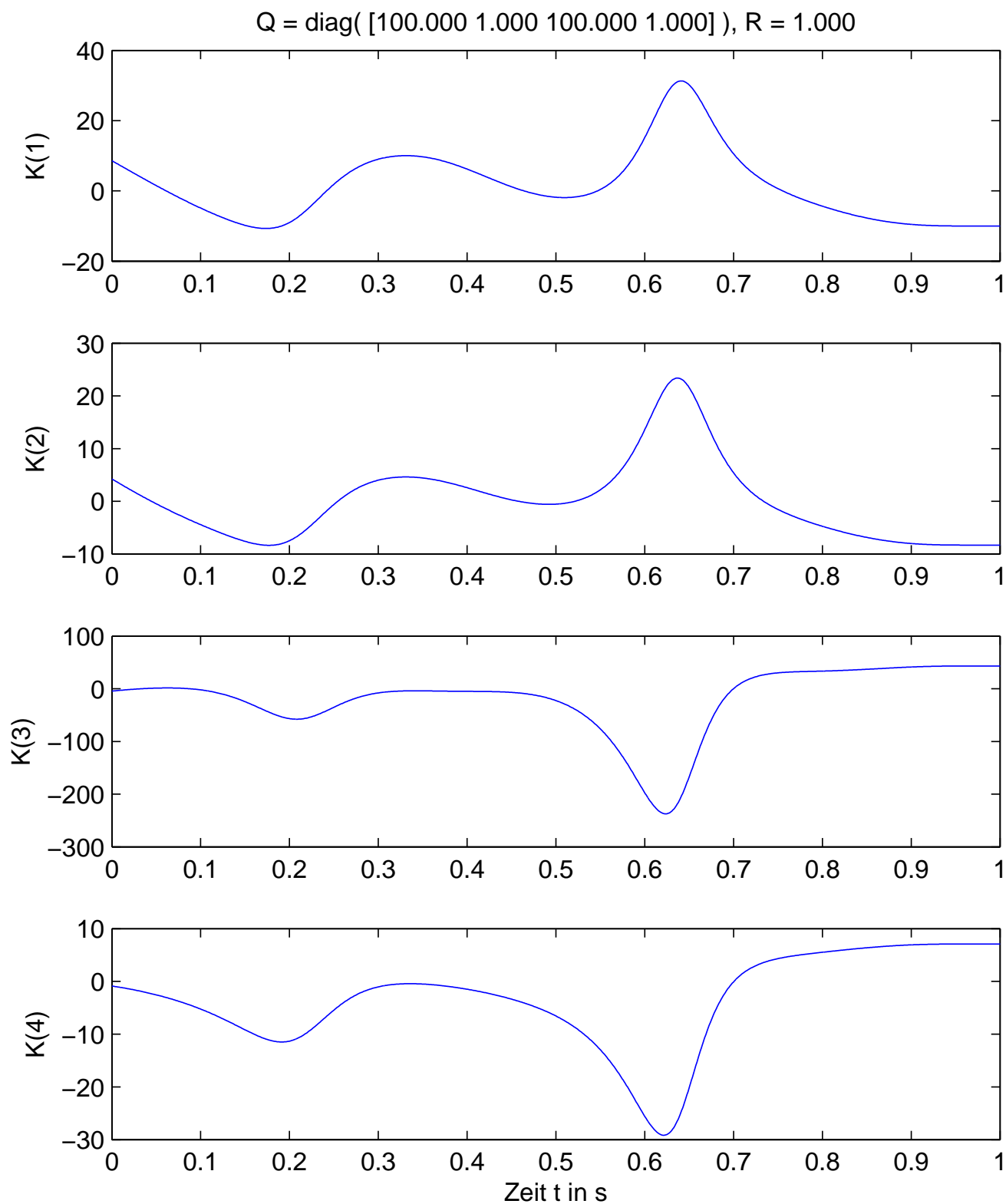


Abbildung 15.2: Verlauf der Reglereinträge für 1 Sekunde

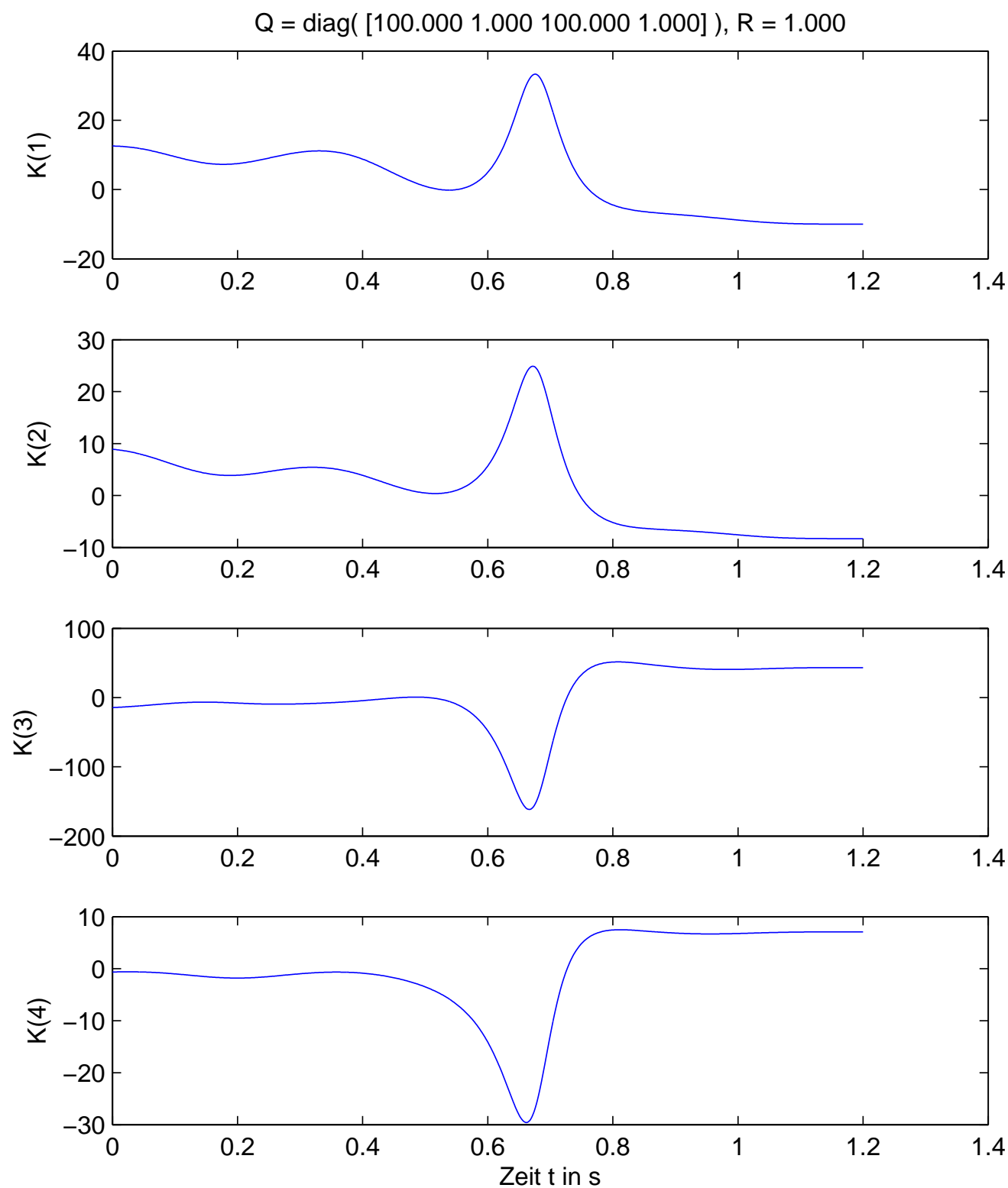


Abbildung 15.3: Verlauf der Reglereinträge für 1,2 Sekunden

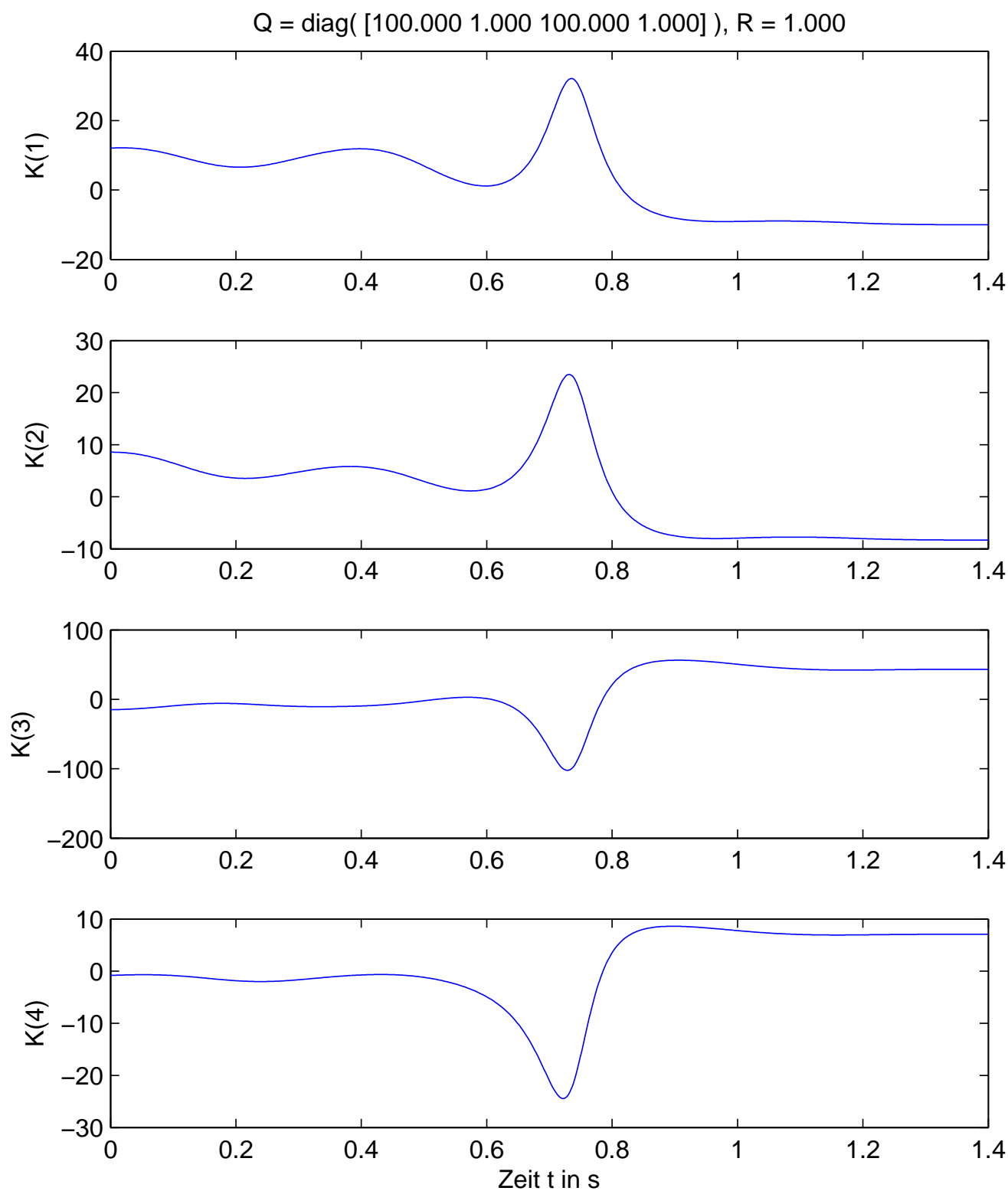


Abbildung 15.4: Verlauf der Reglereinträge für 1,4 Sekunden

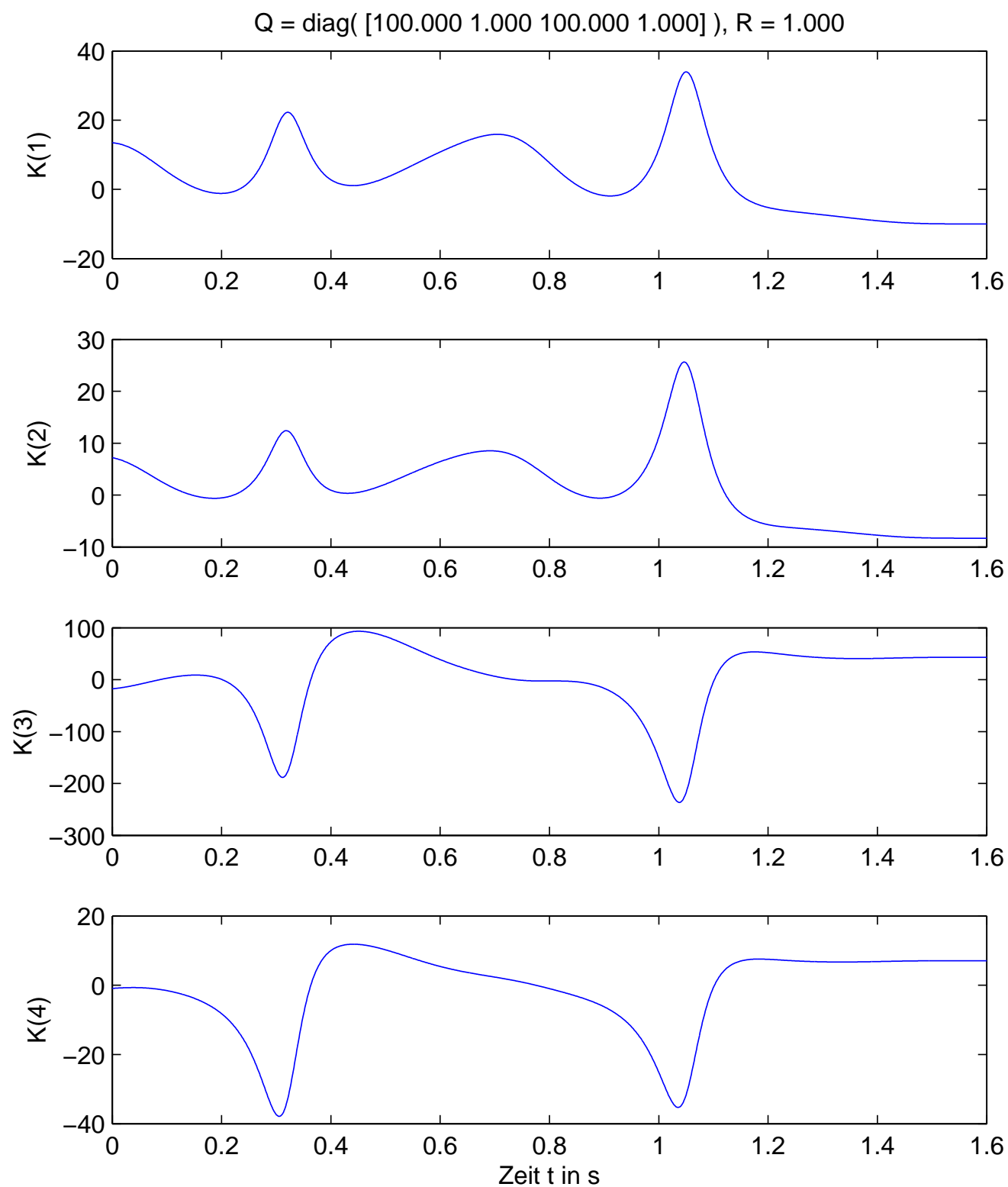


Abbildung 15.5: Verlauf der Reglereinträge für 1,6 Sekunden

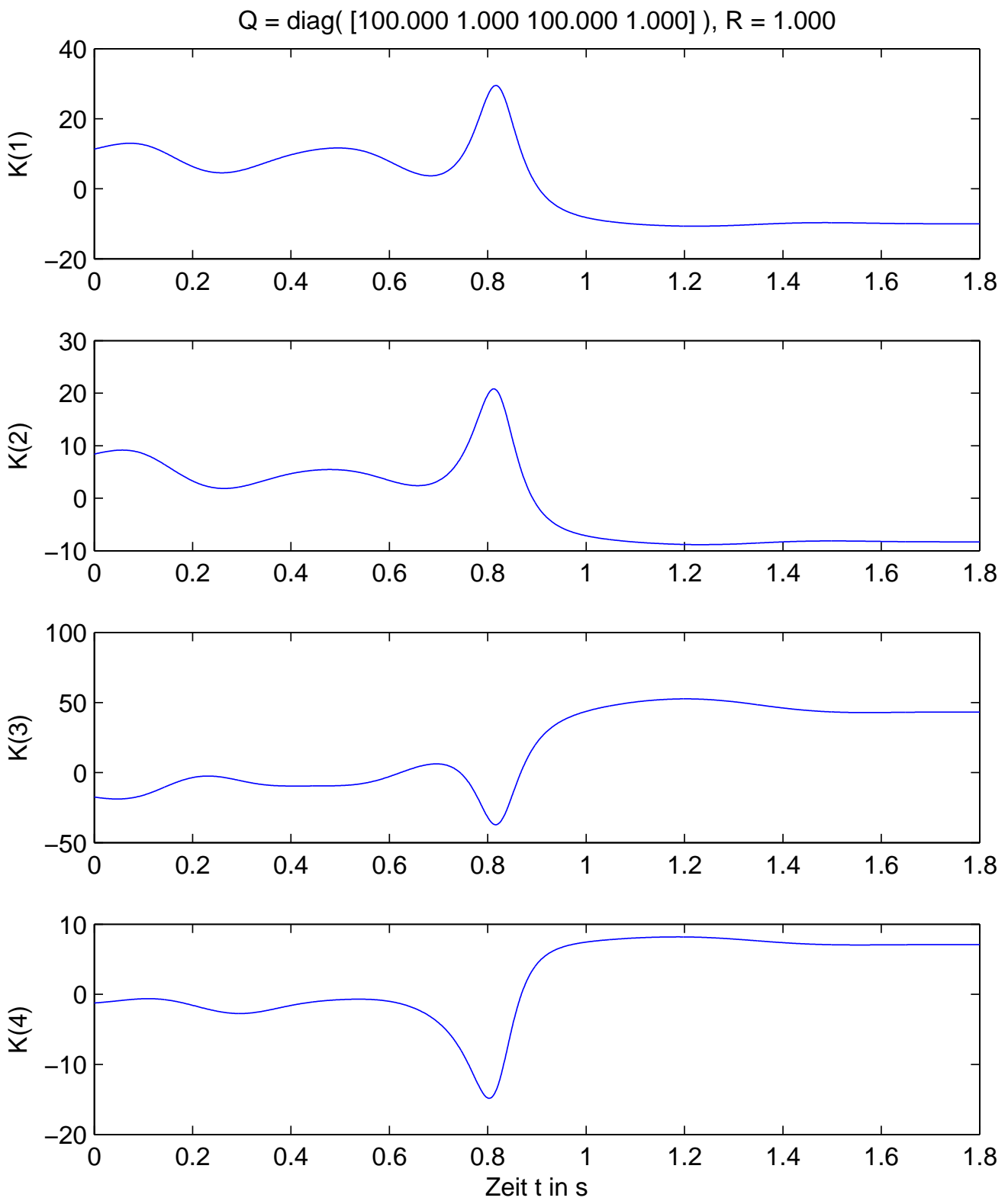


Abbildung 15.6: Verlauf der Reglereinträge für 1,8 Sekunden

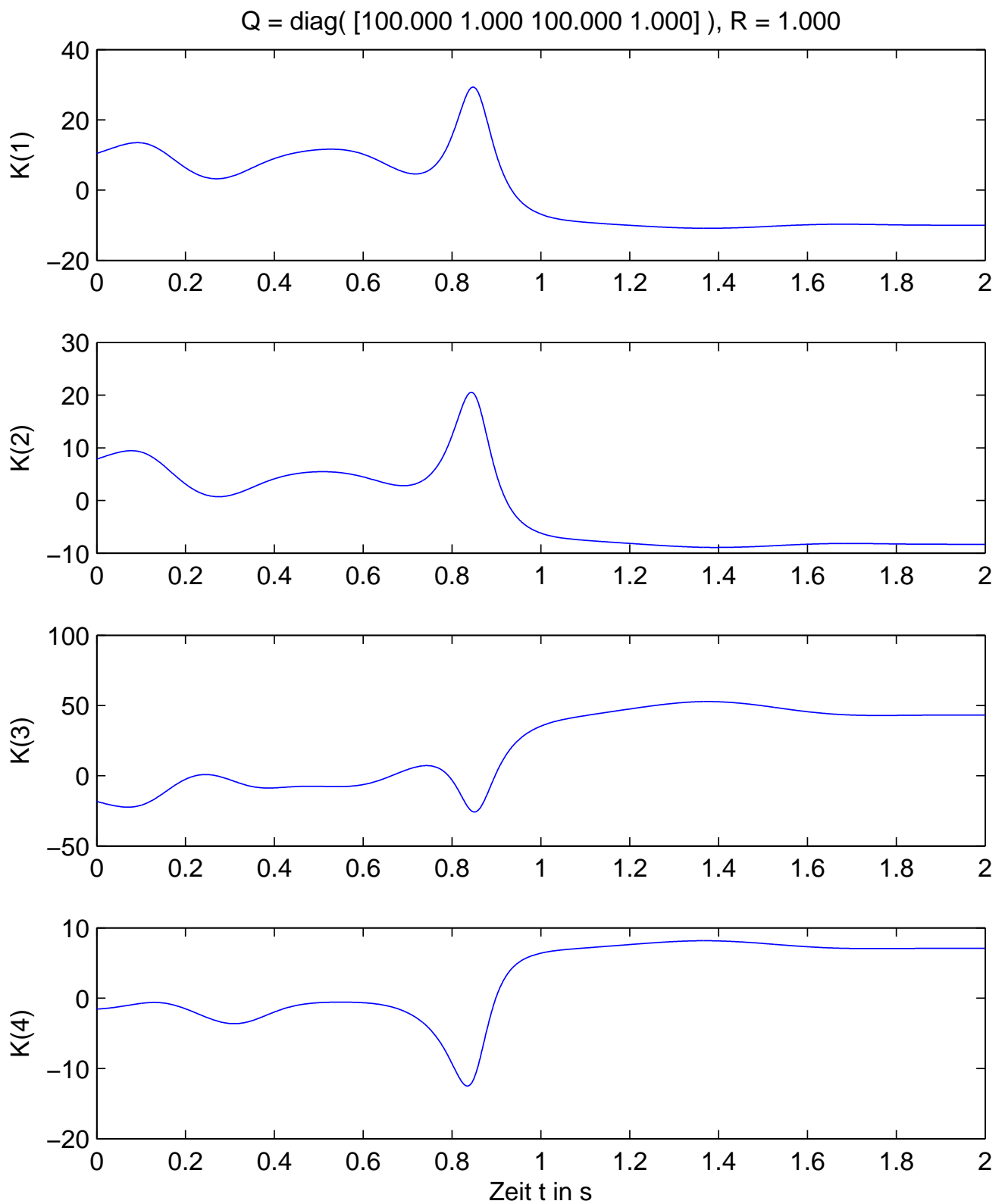


Abbildung 15.7: Verlauf der Reglereinträge für 2 Sekunden

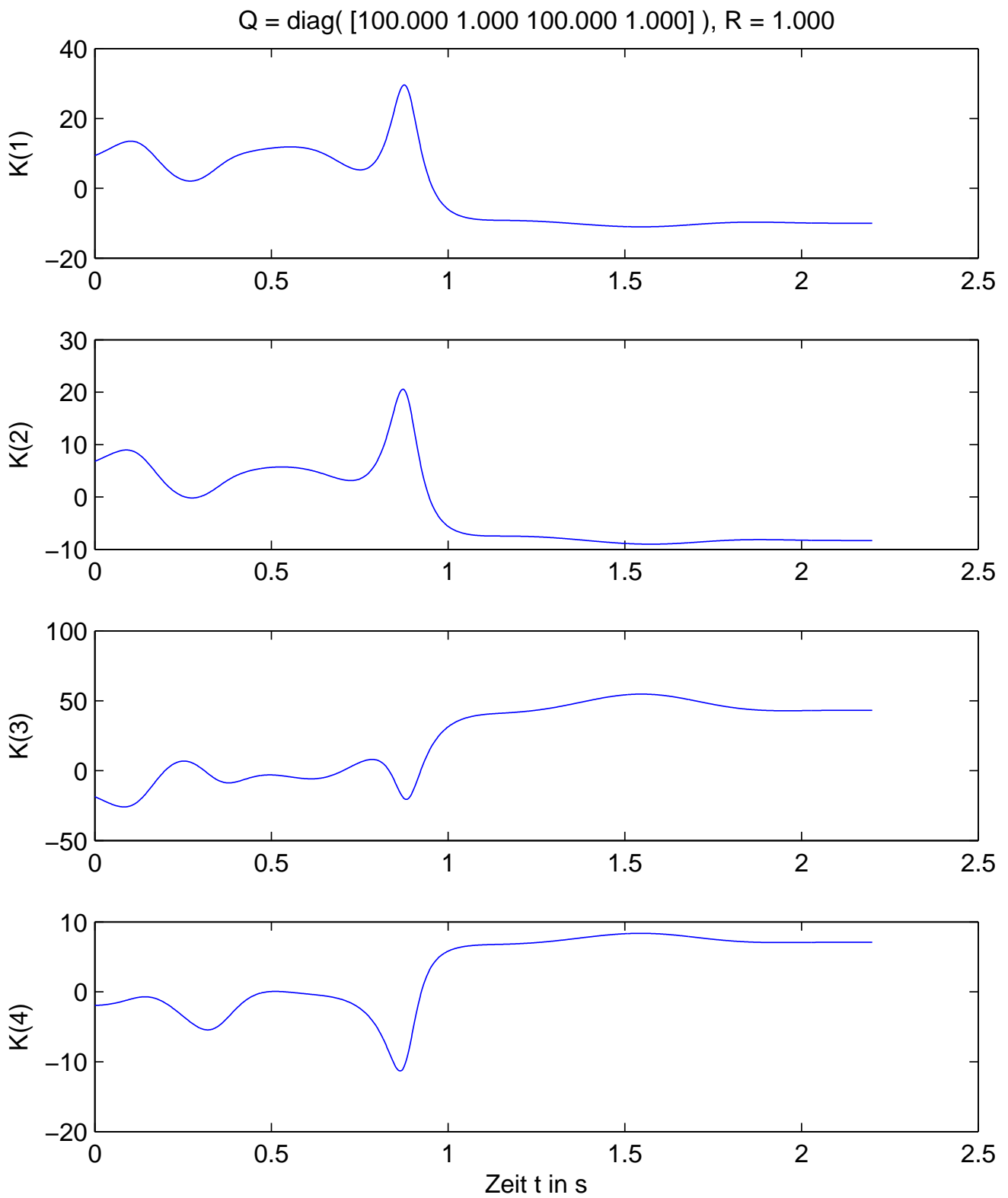


Abbildung 15.8: Verlauf der Reglereinträge für 2,2 Sekunden

16 Auswertung

Grundsätzlich lassen sich diese Ergebnisse deutlich besser an der Animation erkennen. Für alle Simulationen gilt $q_1 = q_3 = 100$ und $q_2 = q_4 = R = 1$.

16.1 Beschleunigungsgesteuertes Modell

Das Pendel erreicht jetzt die obere Ruhelage, auch wenn die Parameter des Pendels von denen, mit denen Steuerung und Regelung berechnet wurden, abweichen. (Im gewissen Rahmen natürlich.) Wenn die obere Lage erreicht ist wird diese auch gehalten. In Abbildung 16.1 sind die Simulationsergebnisse für den Fall dargestellt, dass das Pendel 5 % länger ist. Das unregelte Modell überschlägt sich in diesem Fall, aber das geregelte Modell erreicht eine stabile obere Lage.

16.2 Kraftgesteuertes Modell

Wird die Steuerung mit Folgeregelung auf das Modell angewendet, welches die Kraft auf den Schlitten als Eingang besitzt, dann hat auch eine Veränderung der Pendelmasse einen Einfluss auf das Ergebnis. (In den in Versuch 1 aufgestellten Gleichungen lässt sich die Masse m_p nämlich nicht kürzen, im Gegensatz zu den Gleichungen aus Versuch 5.)

Umgerechnet wird die Beschleunigung in eine Kraft mit

$$F = u \cdot (m_s + m_p).$$

Dies ist allerdings nur eine Näherung, da die durch das bewegte Pendel entstehende Effekte nicht berücksichtigt werden. (Also ist selbst bei genau bekannten Massen diese Formel nicht exakt.) Da aber die Masse des Schlittens groß gegenüber der Pendelmasse ist, sind diese Effekte nur sehr gering und in den Graphen kaum zu sehen, weshalb hier auch keine abgedruckt sind.

Abbildung 16.2 zeigt die Simulationsergebnisse bei kraftgesteuertem Pendel und einer zehnfachen Pendelmasse. Hier tritt schon sehr früh eine deutliche Abweichung des unregulierten Modells in x_1 und x_2 auf und die Endlage wird nicht erreicht, da das Pendel vorher zurückfällt. Das geregelte Modell folgt der Soll-Trajektorie gut. Bei $t = T = 2\text{ s}$ besteht eine kleine Abweichung der Schlittenposition, die im weiteren Verlauf noch ausgeregelt wird.

Nimmt man für den Motor eine Zeitkonstante T_{Motor} von 0,05 s an, so erhält man die in Abbildung 16.3 gezeigten Ergebnisse. Das unregelte Modell folgt der Solltrajektorie zeitversetzt und das Pendel fällt schon bei etwa $t = 1,3\text{ s}$ zurück. Das geregelte Modell beginnt ebenfalls der Trajektorie zeitversetzt zu folgen, holt diesen Rückstand aber schnell auf und erreicht die obere Lage sicher. (Genauer betrachtet entsteht bei jeder Kraftänderung ein geringer Zeitverzug, der dann aber jedesmal schnell aufgeholt wird. Das ist gut an dem Plot mit den Geschwindigkeiten zu erkennen.)

Erhöht man T_{Motor} auf 0,1 s, dann hat die Regelung deutlich mehr Mühe, das Pendel sicher nach oben zu bringen (Abbildung 16.4). Wenn in diesem Fall noch größere Modellunsicherheiten hinzukommen, kann der Trajektorie nicht mehr gefolgt werden.

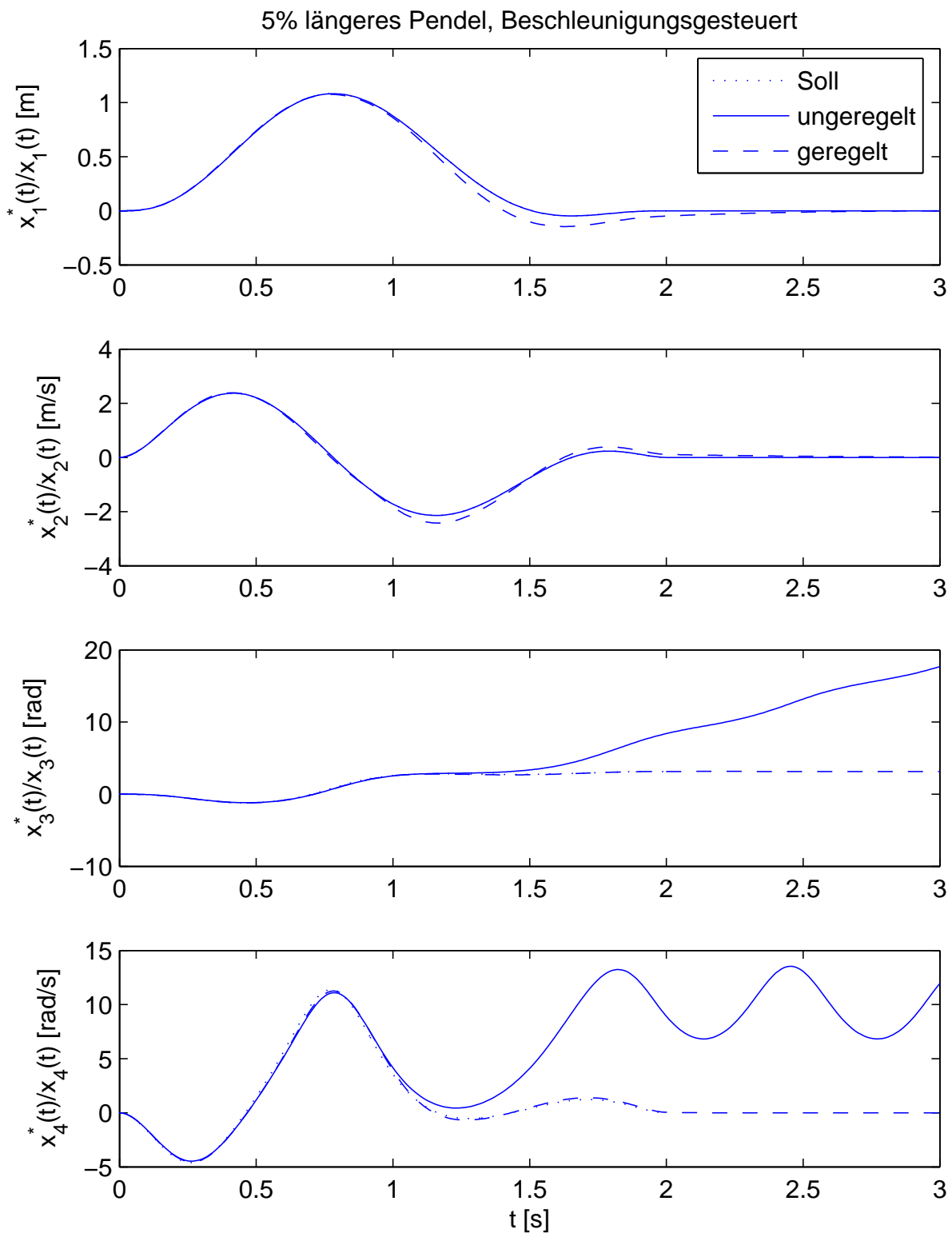


Abbildung 16.1: Simulation mit 5 % längerem Pendel

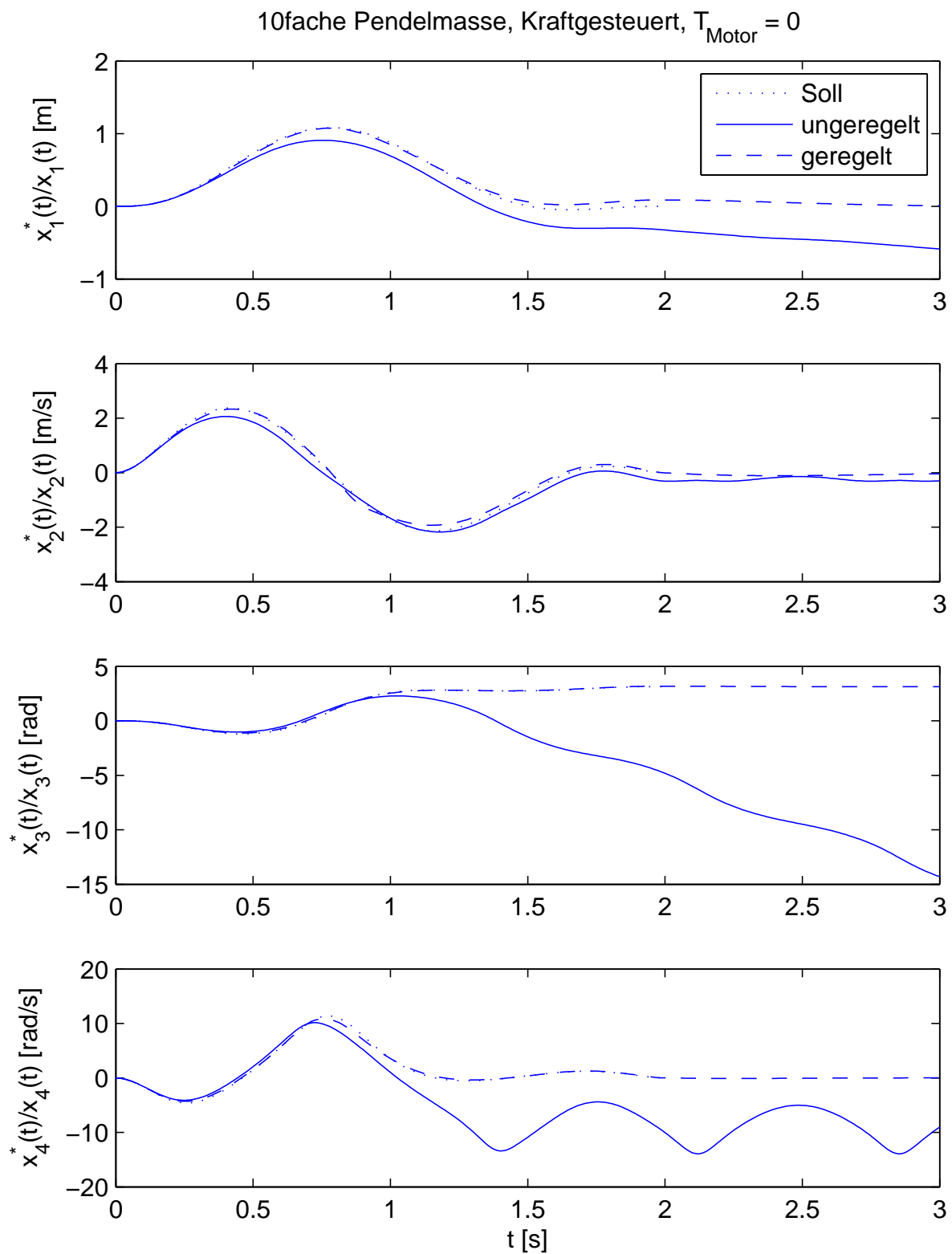


Abbildung 16.2: Simulation mit zehnfach schwererem Pendel, Kraftgesteuert

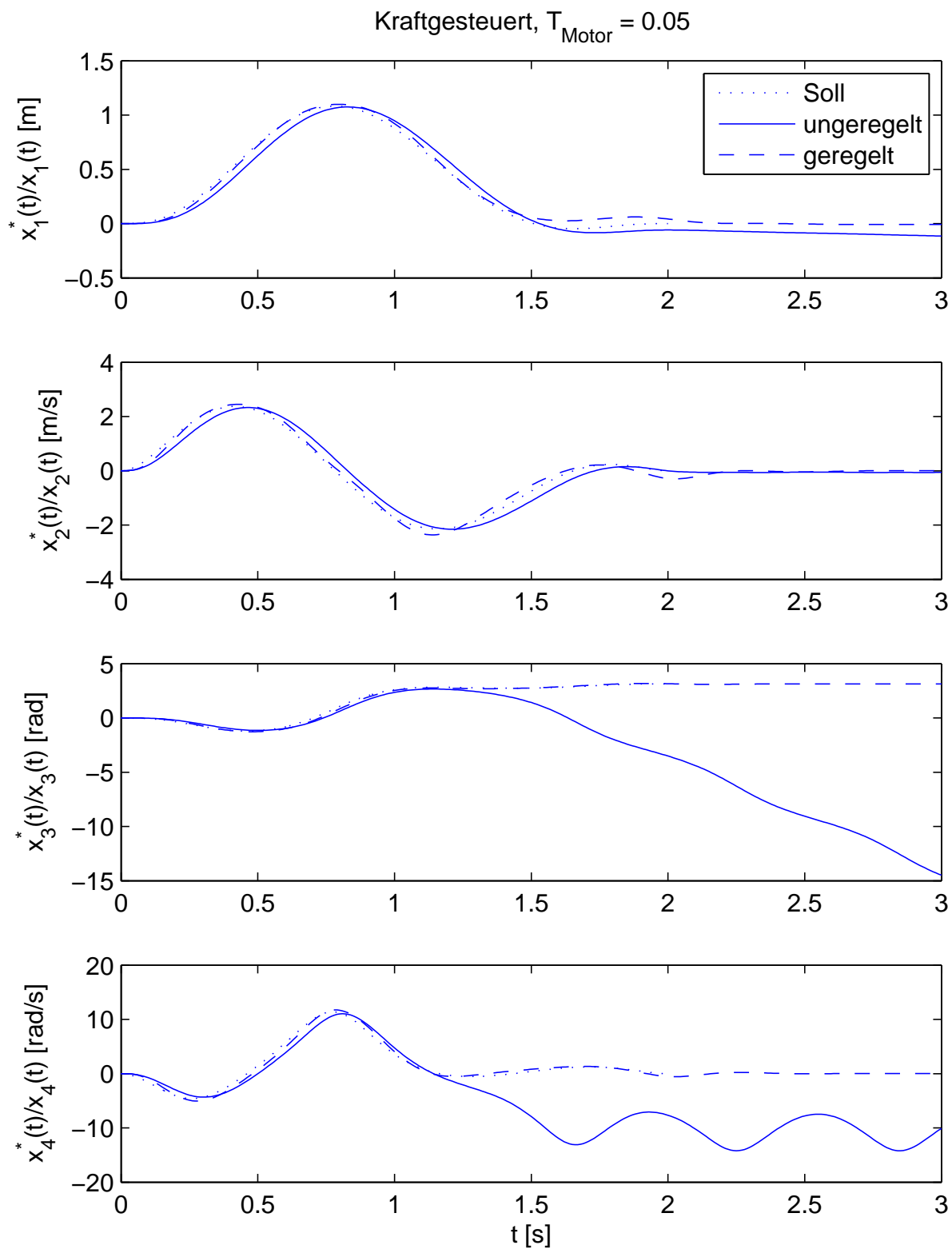


Abbildung 16.3: Simulation mit $T_{\text{Motor}} = 0,05$ s (Kraftgesteuert)

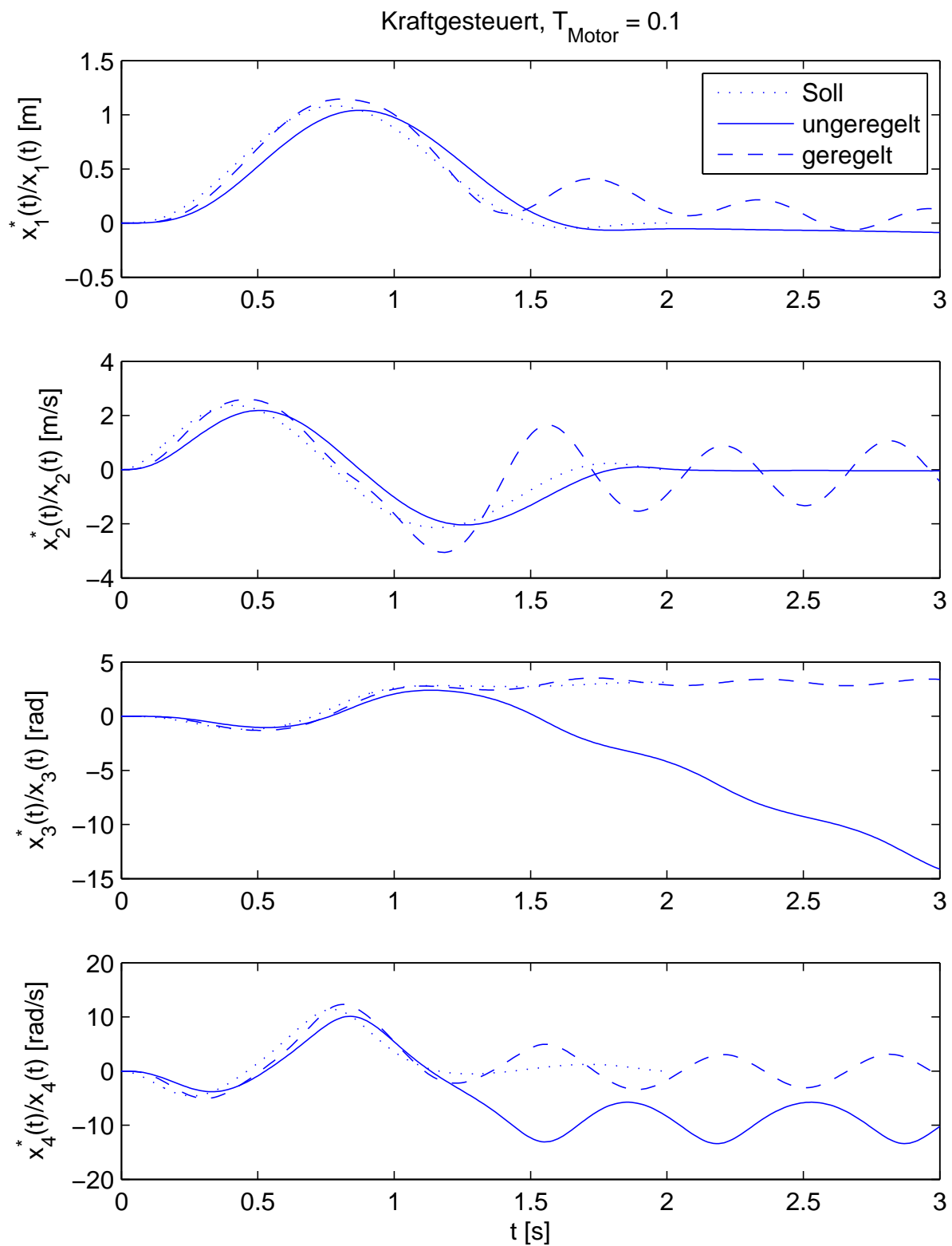


Abbildung 16.4: Simulation mit $T_{\text{Motor}} = 0,1$ s (Kraftgesteuert)

Literaturverzeichnis

- [1] ADAMY, J.: *Systemdynamik und Regelungstechnik II*. Shaker, 2007.
- [2] LUNZE, J.: *Regelungstechnik 1*. Springer, Berlin, 4 Aufl., 2006.
- [3] MARKERT, R.: *Technische Mechanik, Teil B*. Fachbereich Mechanik, Technische Universität Darmstadt, 2002.