2 Systemanalyse mit Matlab/Simulink

In dieser Übungseinheit werden zwei Themenschwerpunkte bearbeitet. Zuerst wird das Computernumerikprogramm MATLAB und die zugehörige Simulationsumgebung SIMULINK vorgestellt. Dieses Softwarepaket wird zur Systemanalyse sowie zur Simulation dynamischer Systeme eingesetzt.

Eine kostenlose MATLAB-Lizenz wird von der TU-Wien zur Verfügung gestellt. Im Computerlabor des Instituts steht MATLAB/SIMULINK in der Version R2020b zur Verfügung. Informationen zum Installieren von MATLAB finden Sie im als QR-Code dargestellten Link. Dieser Link ist im PDF auch anklickbar.

Weiters wird ein Reglerentwurf mittels des Frequenzkennlininenverfahrens durchgeführt. Studieren Sie zur Vorbereitung mindestens folgende Kapitel aus dem Skriptum zur VU Automatisierung (WS 2020/21) [1]:

- Kapitel 3: Lineare Dynamische Systeme,
- Kapitel 4: Der Regelkreis,
- Kapitel 5: Das Frequenzkennlinienverfahren, und
- Kapitel 6: Der Digitale Regelkreis

Alle MATLAB/SIMULINK Dateien, die zum Bearbeiten dieser Übung benötigt werden, finden Sie in `uebung2.zip` auf der Homepage der Lehrveranstaltung.

Bei Fragen oder Anregungen zu dieser Übung wenden Sie sich bitte an

- Lukas Marko <marko@acin.tuwien.ac.at>
- Lukas Tarra <tarra@acin.tuwien.ac.at>.

2.1 Umgang mit Matlab

MATLAB ist ein Computernumerikprogramm. Der Name ist eine Abkürzung für MATrix LABoratory, womit bereits angedeutet wird, dass das Programm ursprünglich für das Rechnen mit Vektoren und Matrizen konzipiert wurde.

Der MATLAB-Desktop (das eigentliche Programmfenster) enthält in der Standardeinstellung folgende Fenster. (Dies kann jedoch individuell angepasst werden.)

- **Command Window**
  Es stellt den Eingabebereich dar, welcher auf jeden Fall angezeigt werden muss. Hier können alle Befehle hinter der Eingabeaufforderung >> eingegeben und direkt ausgeführt werden. Schließt man eine Befehlssequenz mit einem Semikolon ab, so wird die Ausgabe von MATLAB unterdrückt. Das Drücken der Eingabe-Taste bewirkt die sofortige Ausführung der Befehlszeile. Im Falle mehrzeiliger Befehlseingaben kann mit der Umschalt- und der Eingabe-Taste in eine neue Zeile gesprungen werden.

- **Editor**
  Der Editor dient dazu in MATLAB geschriebene Programme (oder andere Textdateien) zu bearbeiten. Matlab-Programme besitzen die Dateiendung `.m` wenn sie als *Matlab Script* oder *Matlab Function* erstellt worden sind, oder die Dateiendung `.mlx` wenn sie als *Live Script* erstellt worden sind. Der Editor erlaubt auch das schrittweise Ausführen und Debuggen von Matlab Scripts.

- **Workspace Browser**
  Die aktuellen Variablen werden in MATLAB im so genannten Workspace angezeigt. Sie können durch Anklicken aufgerufen und verändert werden.

- **Current Folder Browser**
  Im Current Folder Browser wird das aktuelle Arbeitsverzeichnis dargestellt. Dateien und Ordner können geöffnet, angelegt, bearbeitet, etc. werden. Es ist zu empfehlen, dass alle Dateien, auf die während der Rechnung oder Simulation zugegriffen wird, in einem gemeinsamen, lokalen (aktuellen) Verzeichnis liegen.

Aufgabe 2.1. Machen Sie sich mit MATLAB vertraut.

Absolvieren Sie den von MATHWORKS bereitgestellten online-Kurs *MATLAB Onramp*. In diesem wird der Umgang mit MATLAB und die grundlegenden Konstrukte der MATLAB Programmiersprache erklärt. Dazu müssen Sie einen MATHWORKS Account erstellen. Benutzen Sie hierzu ihre TU-Email-Adresse (*...@student.tuwien.ac.at*). Diese kann mit der von der TU Wien zur Verfügung gestellten Lizenz verknüpft werden.

Zusätzlich zu den in *MATLAB Onramp* vorgestellten *Live-Scripts* können MATLAB-Programme auch als *Matlab-Scripts* und *Matlab-Functions* erstellt werden. *Matlab-Scripts* zeichnen sich gegenüber *Live-Scripts* durch schnellere Ausführungszeiten aus, jedoch müssen Plots in extra Fenstern dargestellt werden. *Matlab-Functions* besitzen Übergabewerte und Rückgabewerte. Weiters sind die Variablen innerhalb von *Matlab-Functions* lokal, was bedeutet, dass sie nicht im Workspace aufscheinen.

Auf der Homepage der Lehrveranstaltung finden Sie die *Matlab-Scripts* `cds_matlab_intro_part1.m` und `cds_matlab_intro_part2.m`. Öffnen Sie diese und führen Sie die enthaltenen Befehle schrittweise aus. Beachten Sie, dass die Funktion `mittelwert.m` aufgerufen wird, welche sich daher im aktuellen Arbeitsverzeichnis befinden muss.

Zum Ausführen einzelner Befehlssequenzen markieren Sie diese im Editor und drücken

F9. Zum Ausführen eines ganzen M-files geben Sie entweder dessen Name (ohne Dateiendung) im Command Window ein oder Sie öffnen die Datei im Editor und drücken F5. Versuchen Sie alle Befehle zu verstehen und machen Sie gegebenenfalls von der Hilfefunktion Gebrauch.

Aufgabe 2.2. Gegeben ist das lineare Gleichungssystem

$$\begin{aligned} x_1 + 2x_2 + 4x_3 &= 2 \\ 2x_1 + 2x_2 + x_3 &= 1 \\ 3x_1 + 2x_2 &= 3. \end{aligned} \tag{2.1}$$

Schreiben Sie dieses Gleichungssystem in Matrixdarstellung an und bestimmen Sie den Lösungsvektor $\mathbf{x} = [x_1, x_2, x_3]^{\mathrm{T}}$. Führen Sie die Rechnung einmal mit dem Befehl `inv()` und einmal mit dem Befehl `mldivide()` (oder in seiner Kurzform \) durch. Überlegen Sie sich die Unterschiede der beiden Befehle und wann welcher angewandt werden sollte.

Aufgabe 2.3. Gegeben ist die beim n-ten Summanden abgebrochene Fourier-Reihenentwicklung der Rechteckfunktion

$$\mathrm{rect}(x) \approx A \sum_{k=1}^{n} \frac{4}{\pi(2k-1)} \sin((2k-1)x), \tag{2.2}$$

wobei A die Amplitude bezeichnet. Stellen Sie mit Hilfe einer `for`-Schleife diese Rechteckfunktion für $n = 1, 2, \ldots, 100$ im Intervall $x \in [0, 10]$ dar. Nutzen Sie zur Darstellung der Funktion in der `for`-Schleife den `pause`-Befehl.

Hinweis: Wählen Sie für die Darstellung eine geeignete Schrittweite Δx so, dass das Abtasttheorem erfüllt ist.

Aufgabe 2.4. Gegeben ist das Polynom

$$p_1(s) = s^3 + 3s^2 + 4s + 9, \tag{2.3}$$

und $p_2(s)$ sei das charakteristische Polynom der Matrix

$$\mathbf{A} = \begin{bmatrix} -1 & 2 \\ -3 & 10 \end{bmatrix}. \tag{2.4}$$

1. Berechnen Sie in MATLAB das Polynom $p_3(s) = p_1(s)p_2(s)$.
2. Schreiben Sie eine Funktion mit der Schnittstelle `pd = polydiff(p)`, welche als Argument `p` die Koeffizienten eines beliebigen Polynoms $p(s)$ erhält und

als Rückgabewert `pd` die Koeffizienten des abgeleiteten Polynoms $\mathrm{d}p(s)/\mathrm{d}s$ zurückgibt.

> **Hinweis:** Verwenden Sie nicht den Befehl `polyder()`, sondern implementieren Sie einen eigenen Algorithmus.

3. Schreiben Sie eine Funktion mit der Schnittstelle `s0 = findzero(p,sstart)`, welche als Argument `p` die Koeffizienten eines beliebigen Polynoms $p(s)$ sowie einen Startwert `sstart` erhält. Die Funktion soll mittels des Newton-Verfahrens ausgehend vom Startwert `sstart` eine Nullstelle `s0` des Polynoms $p(s)$ suchen und zurückgeben. Überlegen Sie sich ein geeignetes Abbruchkriterium. Sie können in Ihrem Algorithmus gegebenenfalls die Funktion `polydiff()` verwenden.
4. Testen Sie die Funktion `findzero()` anhand des Polynoms $p_3(s)$. Sie können dazu natürlich $p_3(s)$ zunächst grafisch darstellen.
5. Bestimmen Sie mit dem Befehl `roots()` die Nullstellen von $p_3(s)$.
6. Ist das charakteristische Polynom der Matrix $\mathbf{A}$ ein Hurwitzpolynom?

> **Hinweis:** Polynome können in MATLAB als Vektoren der absteigend geordneten Polynomkoeffizienten dargestellt werden, d. h. $p(s) = a_n s^n + a_{n-1} s^{n-1} + \ldots + a_1 s + a_0$ wird als Vektor $\mathbf{p} = [a_n, a_{n-1}, \ldots, a_1, a_0]^{\mathrm{T}}$ eingegeben. Zur Auswertung eines Polynoms an einer bestimmten Stelle s kann der Befehl `polyval()` verwendet werden. Zur Bestimmung des charakteristischen Polynoms einer quadratischen Matrix kann der Befehl `poly()` verwendet werden. Die Multiplikation zweier Polynome entspricht der Faltung ihrer Koeffizientenvektoren. Die (diskrete) Faltungsoperation kann mit dem Befehl `conv()` durchgeführt werden.

2.1.1 Control System Toolbox

Toolboxen sind Sammlungen von Funktionen, die den Funktionsumfang von MATLAB erweitern. Nach der erstmaligen Installation werden Toolboxen automatisch beim Starten von MATLAB geladen. Eine Übersicht über die installierten Toolboxen erhält man mit dem Kommandozeilenbefehl `ver`. Die *Control System Toolbox* ist häufig bei regelungstechnischen Aufgabenstellungen nützlich. Sie unterstützt bei der Analyse und dem Reglerentwurf von linearen dynamischen Systemen.

Aufgabe 2.5. Öffnen Sie die Datei `cds_matlab_intro_part3.m` im MATLAB-Editor und arbeiten Sie alle Befehle schrittweise durch. Versuchen Sie alle Befehle zu verstehen und machen Sie gegebenenfalls von der Hilfefunktion Gebrauch.

Das Bestimmen des Verhaltens eines Systems auf einen Eingang (z. B. die Sprungantwort)

wird mathematisch als Anfangswertproblem

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}, t)\,, \qquad \mathbf{x}(t_0) = \mathbf{x}_0 \tag{2.5a}$$
$$\mathbf{y} = \mathbf{h}(\mathbf{x}, \mathbf{u}, t) \tag{2.5b}$$

formuliert. Anfangswertprobleme können mit numerischen Integrationsalgorithmen gelöst werden. Die Funktion eines solchen Integrationsalgorithmus soll anhand des expliziten Euler-Verfahrens (Euler-vorwärts-Verfahren) kurz verdeutlicht werden. Beim Euler-vorwärts-Verfahren wird die zeitliche Ableitung mittels des Differenzenquotienten

$$\dot{\mathbf{x}}(T_a k) \approx \frac{\mathbf{x}((k+1)T_a) - \mathbf{x}(kT_a)}{T_a} \tag{2.6}$$

mit der Schrittweite T_a angenähert. Dadurch ergibt sich als Approximation der Lösung des Anfangswertproblems (2.5b)

$$\mathbf{x}((k+1)T_a) = \mathbf{x}_{k+1} \approx \mathbf{x}_k + T_a \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k, kT_a)\,, \qquad \mathbf{x}_0 = \mathbf{x}(t_0)\ . \tag{2.7}$$

Aufgabe 2.6. In dieser Aufgabe soll die Sprungantwort des Systems

$$\dot{x}_1 = x_2, \qquad x_1(0) = 0 \tag{2.8a}$$
$$\dot{x}_2 = -x_1 - 2x_2 + u, \qquad x_2(0) = 0 \tag{2.8b}$$
$$y = x_1 \tag{2.8c}$$

für den Zeitraum $t = 0, \ldots, 10\,\mathrm{s}$ durch Simulation bestimmt werden.

1. Lösen Sie das Anfangswertproblem (2.8) mit Hilfe des Euler-vorwärts-Verfahrens mit der Schrittweite $T_a = 0.5\,\mathrm{s}$ und stellen Sie die berechnete Sprungantwort grafisch dar.

2. Lösen Sie das Anfangswertproblem (2.8) mit Hilfe des MATLAB-Befehls `ode45` und stellen Sie die berechnete Sprungantwort grafisch dar. Beachten Sie hierbei, dass die Differentialgleichung (ODE) als *Function Handle* übergeben werden muss.

3. Berechnen Sie unter Verwendung des Befehls `step()` der Control System Toolbox die Sprungantwort des kontinuierlichen Systems (2.8) und stellen Sie die berechnete Sprungantwort grafisch dar.

4. Tasten Sie das System (2.8) mittels des Befehls `c2d()` (Halteglied nullter Ordnung) und einer Abtastzeit von $T_a = 0.5\,\mathrm{s}$ ab und stellen Sie die berechnete Sprungantwort grafisch dar. Um zeitlich abgetastete Signale dar zu stellen ist der Befehl `stairs` geeignet.

5. Wie unterscheiden sich die Ergebnisse?

2.1.2 Simulink

SIMULINK ist eine Erweiterung von MATLAB zur Simulation und Analyse dynamischer Systeme. Mithilfe von SIMULINK gestaltet sich das Lösen von Anfangswertproblemen einfach. Die grafische Bedienoberfläche erlaubt die Beschreibung von dynamischen Systemen mithilfe von Blockschaltbildern. Hierbei stellt SIMULINK eine Bibliothek mit vorgefertigten Funktionsblöcken zur Verfügung, welche durch benutzerdefinierte Blöcke erweitert werden können.

Aufgabe 2.7. Machen Sie sich mit Simulink vertraut.

Absolvieren Sie den in SIMULINK bereitgestellten Kurs Simulink Onramp. Starten Sie hierzu SIMULINK mittels dem MATLAB-Befehl `simulink`. In dem sich öffnenden Fenster klicken Sie links unten auf *Simulink Onramp* um den Onramp - Kurs zu starten. Weitere Informationen finden Sie unter dem Link hinter dem QR-Code.

Zum effizienten Arbeit mit SIMULINK wird folgendes Vorgehen empfohlen:

1. Parameterwerte werden nicht direkt in SIMULINK eingetragen, sondern zusammengefasst in einem Matlab-Script definiert, welches vor der Simulation ausgeführt wird. Damit können die Parameter einfach und an zentraler Stelle geändert werden. Zum Update der Parameter muss das Matlab-Script erneut ausgeführt werden.

   Hinweis: Alle Variablen, die sich im Matlab-Workspace befinden, stehen auch in SIMULINK zur Verfügung.

2. Werden die mathematischen Ausdrücke umfangreicher, empfiehlt es sich, die Verschaltung vieler Einzelblöcke durch die Verwendung benutzerdefinierter (programmierter) Blöcke zu umgehen. Die entsprechenden Blöcke befinden sich in der Gruppe `User-Defined Functions`. Im Block `Fcn` können sowohl MATLAB Funktionen als auch benutzerdefinierte Funktionen verwendet werden.

3. Eine weitere Möglichkeit die Übersichtlichkeit von Modellen zu verbessern, ist die Verwendung von Subsystemen (`Ports & Subsystems` → `Subsystem`).

4. Um die Anzahl der am Bildschirm dargestellten Signalflussleitungen zu reduzieren, können die Blöcke `From` und `Goto` aus der Gruppe `Signal Routing` verwendet werden.

5. LTI-Systeme, welche mittels der Control System Toolbox als Übertragungsfunktion oder in Zustandsraumdarstellung erzeugt wurden, können direkt in Simulink eingebunden werden (`Control System Toolbox` → `LTI System`).

Nach der Erstellung eines Modells (Blockschaltbild) sind die Simulationseinstellungen vorzunehmen, was im Menüpunkt *Model Settings* im Reiter *Modeling* erfolgen kann. Wesentlich ist dabei die Wahl der Simulationsdauer und des Integrationsalgorithmus (Solver). Der momentan eingestellte Solver wird rechts in der Fußzeile des Simulinkmodells angezeigt. Auch durch klicken auf diesen kann man die Simulationseinstellungen

öffnen. Ferner können für den Integrationsalgorithmus Schranken der Zeitschrittweite und Genauigkeitsanforderungen eingestellt werden. Im Rahmen dieser Einführung soll auf eine detaillierte Diskussion der verwendeten Algorithmen verzichtet werden. Einen Überblick über einige in MATLAB zur Verfügung stehende Solver für Anfangswertprobleme erhalten Sie in der Hilfe *ode23, ode45, ode113, ode15s, ode23s, ode23t, ode23tb* (aufrufbar z. B. mit `doc ode45`) unter der Überschrift *Algorithms*. Diese Algorithmen werden, neben anderen, auch von SIMULINK verwendet. Ferner sei auf die Fachliteratur, z. B. [2, 3], verwiesen.

Aufgabe 2.8. Machen Sie sich weiter mit Simulink vertraut.

Auf der Homepage der Lehrveranstaltung finden Sie das Simulink-Modell `cds_simulink_intro_part1.slx`.

Öffnen Sie dieses Modell und starten Sie die Simulation. Achten Sie darauf, zuvor das *Matlab-Script* `init_cds_simulink_intro_part1.m` auszuführen. Dieses *Matlab-Script* erstellt alle für die Simulation notwendigen Variablen im MATLAB-Workspace. Versuchen Sie alle Blöcke zu verstehen und machen Sie gegebenenfalls von der Hilfefunktion Gebrauch.

Implementierung von zeitkontinuierlichen dynamischen Systemen mittels einer Integratorkette

Um eine Differentialgleichung höherer Ordnung mit Simulink lösen zu können, muss diese als System von Differentialgleichungen erster Ordnung dargestellt werden. Dieses Vorgehen wird in Beispiel 2.1 dargestellt.

Beispiel 2.1. Das Anfangswertproblem

$$\dddot{y} + \cos^2(y)\ddot{y} + a\dot{y} + u = 0\,, \qquad \ddot{y} = \dot{y} = y = 0\,, \qquad u = \sin(t) \tag{2.9}$$

mit einem Parameter $a = 0.3$ soll mittels einer Integratorkette dargestellt werden. Zuerst wird die Differentialgleichung in Zustandsraumdarstellung $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, u)$ übergeführt. Da (2.9) eine Differentialgleichung dritter Ordnung ist, wird ein dreidimensionaler Zustand

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} y \\ \dot{y} \\ \ddot{y} \end{bmatrix} \tag{2.10}$$

verwendet. Damit ergibt sich ein System von drei Differentialgleichungen erster Ordnung

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} x_2 \\ x_3 \\ -\cos^2(x_1)x_3 - ax_2 - u \end{bmatrix}, \qquad \mathbf{x}(0) = \begin{bmatrix} x_1(0) \\ x_2(0) \\ x_3(0) \end{bmatrix}. \tag{2.11}$$

Da man an der Lösung $\mathbf{x}(t)$ interessiert ist, wird (2.11) einmal zeitlich integriert. Dadurch ergibt sich

$$\mathbf{x}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \end{bmatrix} = \begin{bmatrix} x_1(0) + \int_0^t x_2(\tau)\,\mathrm{d}\tau \\ x_2(0) + \int_0^t x_3(\tau)\,\mathrm{d}\tau \\ x_3(0) + \int_0^t (-\cos^2(x_1(\tau))x_3(\tau) - ax_2(\tau) - u(\tau))\,\mathrm{d}\tau \end{bmatrix} \tag{2.12}$$

In (2.12) ist ersichtlich, dass drei Integratoren benötigt werden. Jeder dieser Integratoren kann mittels eines Integratorblocks in Simulink dargestellt werden. Damit kann das Differentialgleichungssystem in Simulink wie in Abbildung 2.1 dargestellt werden.

Hinweis: Die Anfangszustände $(x_{0,1}, x_{0,2}, x_{0,3})$ müssen in den Einstellungen der einzelnen Integratorblöcke definiert werden.

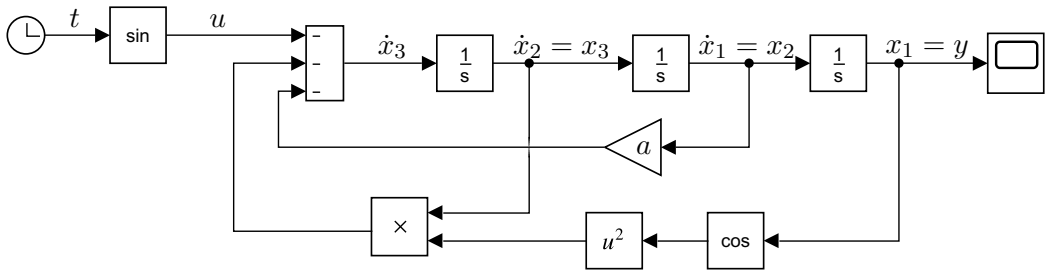

Abbildung 2.1: Implementierung von (2.12) mittels einer Integratorkette.

Implementierung von zeitkontinuierlichen dynamischen Systemen mittels *Matlab Function*-Blöcken.

Die Implementierung einer Differentialgleichung mittels Integratorkette wird bei hochdimensionalen Problemen und steigender mathematischer Komplexität unübersichtlich. Durch das Benutzen von *Matlab Function*-Blöcken kann dies verhindert werden. Das Benutzen von *Matlab Function*-Blöcken wird in Beispiel 2.2 erläutert.

Beispiel 2.2. Das Anfangswertproblem (2.9) soll mittels eines *Matlab Function*-Blocks gelöst werden. Wird das System in Zustandsraumdarstellung $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, u)$ übergeführt, vgl. (2.11). Durch Integration ergibt sich dann

$$\mathbf{x}(t) = \mathbf{x}_0 + \int_0^t \mathbf{f}(\mathbf{x}(\tau), u(\tau))\,\mathrm{d}\tau \tag{2.13a}$$

$$y(t) = h(\mathbf{x}(t), u(t)). \tag{2.13b}$$

Diese Darstellung kann direkt als *Matlab Function* in Simulink implementiert werden. Dazu wird in Simulink ein *Matlab Function*-Block (`User Defined Functions` → `Matlab Function`) eingefügt. Durch Doppelklick öffnet sich ein Fenster im Matlab

Editor, in welchem programmiert werden kann.

Listing 2.1: Implementierung von (2.13).

```
function [xpunkt, y] = fcn(u,xx,param)
        % Aufteilen der Parameter (zur besseren Lesbarkeit)
        a = param.a;

        % Aufteilen des Zustandsvektors (zur besseren Lesbarkeit)
        x1 = xx(1);
        x2 = xx(2);
        x3 = xx(3);

        % Implementierung von f(x,u)
        f1 = x2;
        f2 = x3;
        f3 = -cos(x1)^2 * x3 - a * x2 - u;

        % Zuweisen entsprechend xpunkt = f(x,u)
        xpunkt = [f1; f2; f3];

        % Implementierung von h(x,u)
        y = x1;
end
```

Die Darstellung in (2.13) kann dann z. B. mittels des Programmcodes in Listing 2.1 implementiert werden. Hierbei besitzt die Funktion `fcn` drei Übergabeparameter (`xx`, `u`, `param`) und zwei Rückgabewerte (`xpunkt`, `y`). Ist der *Matlab Function*-Block im Matlab Editor geöffnet, so kann man im Reiter `Editor` mit dem Button `Edit Data` den `Ports and Data Manager` öffnen. In diesem kann eingestellt werden ob ein Übergabeparameter als Eingang oder als Parameter behandelt werden soll. Parameter müssen im Matlab-Workspace definiert werden. Es ist empfehlenswert alle benötigten Parameter in einem *Struct Array* zu organisieren. Hierbei ist es wichtig, dass im *Ports and Data Manager* eingestellt wird, dass der Parameter nicht *Tunable* ist.

Die Integration von `xpunkt` wird nun mittels eines Integratorblocks in Simulink durchgeführt. Dieser Block kann auch mit vektorwertigen Signalen umgehen. Dabei ist zu beachten, dass die Dimension des Anfangszustandes des Integratorblocks der Dimension des Zustandes des Systems entsprechen muss.

Abschließend zeigt Abbildung 2.2 die vollständige Implementierung von (2.13) mittels *Matlab Function*-Blöcken.

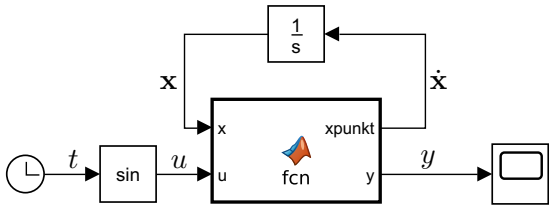

Abbildung 2.2: Implementierung von (2.12) mittels einer Matlab Function.

Implementierung von zeitdiskreten dynamischen Systemen mittels *Matlab Function*-Blöcken

Auch zeitdiskrete Systeme können mittels *Matlab Function*-Blöcken implementiert werden. Hierbei gibt es mehrere Möglichkeiten. Die im Folgenden erklärte Methode bietet den Vorteil, dass der resultierende Programmcode direkt in einem Digitalrechner implementiert werden kann. Dadurch können entworfene Regler oder Beobachter mit geringem Zeitaufwand auf industrielle Anlagen oder auf Versuchsaufbauten portiert werden. Im Speziellen ist es auf einfache Weise möglich die Matlab-Function direkt in der Rapid Prototyping System DSPACE zu verwenden. Im nachfolgendem Beispiel wird erklärt wie ein zeitdiskretes System mittels *Matlab Function*-Blöcken in Simulink implementiert werden kann.

Beispiel 2.3. Das Anfangswertproblem (2.9) soll mit Hilfe des Euler-vorwärts-Verfahrens als zeitdiskretes System mit der Abtastzeit $T_a = 100\,\text{ms}$ simuliert werden. Zuerst wird analog zu Beispiel 2.2 das System in Zustandsraumdarstellung übergeführt und zeitlich integriert (vgl. (2.13)). Das Integral (2.13) wird dabei über das Euler-vorwärts-Verfahrens (2.7) approximiert. Daraus ergibt sich

$$\mathbf{x}_{k+1} = \mathbf{x}_k + T_a \mathbf{f}(\mathbf{x}_k, u_k)\,, \qquad \mathbf{x}_0 = \mathbf{x}(0) \tag{2.14a}$$

$$y_{k+1} = h(\mathbf{x}_k, u). \tag{2.14b}$$

Diese Darstellung kann direkt in einer *Matlab Function* in Simulink implementiert werden. Der entsprechende Programmcode ist in Listing 2.2 angegeben. Hierbei wird der Zustand `xx` als *persistent* Variable definiert, welche über mehrere Funktionsaufrufe erhalten bleiben. Beim Erstellen dieser Variable ist diese leer (das bedeutet das Gleiche wie die Zuweisung `xx=[]`). Man beachte, dass im ersten Aufruf des *Matlab Function*-Blocks die Variable `xx` mit dem Anfangswert `xx0` initialisiert wird.

Listing 2.2: Implementierung von (2.14).

```
function y = fcn(u,param)
        % Aufteilen der Parameter
        a = param.a;
        Ta = param.Ta;

        % Definition des diskreten Zustandes als persistent Variable
        persistent xx;

        % Initialisierung des diskreten Zustandes
        if isempty(xx); xx=param.xx0; end

        % Aufteilen des Zustandsvektors
        x1 = xx(1);
        x2 = xx(2);
        x3 = xx(3);

        % Implementierung von f(x,u)
        f1 = x2;
        f2 = x3;
```

```
        f3 = -cos(x1)^2*x3 -a*x2 -u;

        % Zusammenfuegen zu einem Vektor
        ff = [f1; f2; f3];

        % Approximation mittels Euler-Verfahren
        xx = xx + Ta*ff;

        % Implementierung von h(x,u)
        y = x1;
end
```

Für die Implementierung eines zeitdiskreten Systems in SIMULINK ist es notwendig die Abtastzeit zu spezifizieren. Dazu wird im *Ports and Data Manager* die *Update Method* als *discrete* festlegt, siehe dazu Abbildung 2.3.

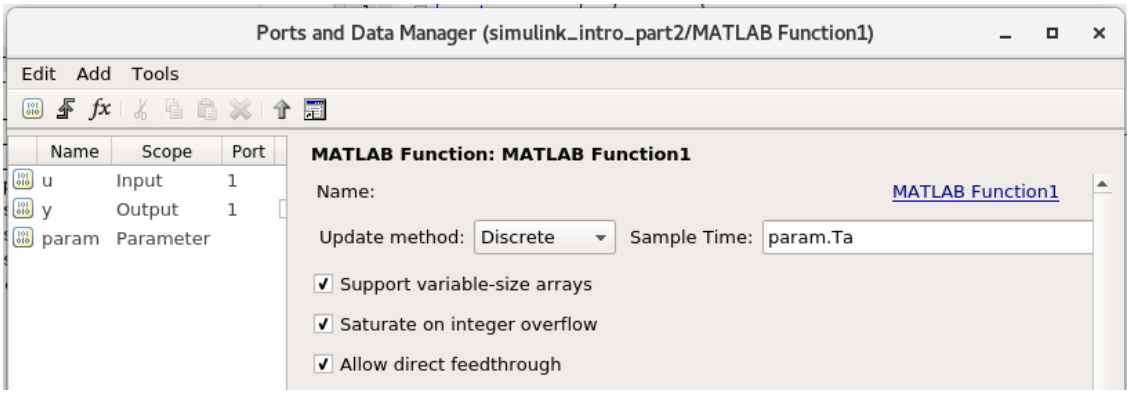

Abbildung 2.3: Einstellungen im *Ports and Data Manager* für zeitdiskrete Systeme.

Aufgabe 2.9. Machen Sie sich mit der Implementierung dynamischer Systeme vertraut.

Auf der Homepage der Lehrveranstaltung finden Sie das Simulink-Modell `simulink_intro_part2.slx`. In diesem Simulinkmodell sind Beispiel 2.1, Beispiel 2.2 und Beispiel 2.3 implementiert.

Öffnen Sie dieses Modell und starten Sie die Simulation. Achten Sie darauf zuvor das Matlab-Script `init_cds_simulink_intro_part2.m` auszuführen. Dieses *Matlab Script* erstellt alle für die Simulation notwendigen Variablen im *Matlab Workspace*. Versuchen Sie alle Blöcke zu verstehen und machen Sie gegebenenfalls von der Hilfefunktion Gebrauch. Achten Sie besonders auf die Einstellungen die im *Ports and Data Manager* getroffen wurden.

Aufgabe 2.10.

1. Implementieren Sie das mit dem Befehl `c2d()` diskretisierte System (Abtastzeit

$T_a = 0.1s$) aus Aufgabe 2.6 in Form eines *Matlab Function*-Blocks.

2. Implementieren Sie das System außerdem mittels eines LTI-Blocks und vergleichen Sie die Ergebnisse für verschiedene Eingangsfunktionen.

2.2 Rotary Flexible Joint

2.2.1 Simulation

Im Weiteren soll das Modell für den Rotary Flexible Joint (RFJ), welches in der vorherigen Übung hergeleitet wurde, in Simulink implementiert werden. Führen Sie dazu folgende Aufgaben durch.

Aufgabe 2.11. Implementieren Sie das nichtlineare Modell (1.17) aus Aufgabe 1.8 mittels *Matlab Function*-Blöcken in SIMULINK. In Abbildung 2.4 ist die gewünschte Struktur der Implementierung dargestellt. Abbildung 2.5 zeigt wie die Anfangswerte an den Integratorblock übergeben werden können. Verwenden Sie als Eingang des *Matlab Function*-Blocks das Motormoment τ_{mot}, die Störkraft f_s sowie die Zustände $\mathbf{x} = [\varphi_a, \omega_a, \varphi_t, \omega_t]^{\mathrm{T}}$ und als Ausgang die zeitlichen Ableitungen der Zustände $\dot{\mathbf{x}}$. Alle physikalischen Systemparameter sollen mittels eines *Struct arrays* als Parameter an den *Matlab Function*-Block übergeben werden. Als gemessener Ausgang des Systems soll, gemäß (1.17), der Absolutwinkel des Auslegers $y = \varphi_a$ zur Verfügung stehen. Untersuchen sie das Systemverhalten anhand eines rechteckförmigen Eingangsmoments und überprüfen Sie dieses auf Plausibilität.

Zum Test und zum Abgleich Ihres Modells steht auf der Homepage der Lehrveranstaltung eine `p-function` des Systems Rotary Flexible Joint zum Download zur Verfügung (`RFJ_Student_S.p`). Die Datei `Simulation_RFJ_Student.slx` zeigt wie diese `p-function` in ein Simulinkmodell eingebunden werden kann.

Hinweis: Alle Parameter und Anfangsbedingungen können in einer einzigen Struktur (`struct`) zusammengefasst an den *Matlab Function*-Block bzw. dem Integrator übergeben werden. Innerhalb des *Matlab Function*-Blocks kann wie gewohnt auf die Einträge dieser Struktur zugegriffen werden. Die Einträge für diese Struktur finden Sie in der Matlab Datei `RFJ_Parameter_Student.m`.

Hinweis: Man kann in MAPLE berechnete analytische Ausdrücke mit dem Befehl `Matlab` welcher im MAPLE-Paket `CodeGeneration` enthalten ist direkt in Matlabprogrammcode umwandeln. Dieser Programmcode muss nur mehr von MAPLE in `Matlab` kopiert werden.

Aufgabe 2.12. Implementieren Sie das linearisierte Modell des Rotary Flexible Joint aus Aufgabe 1.10 in Form seiner Zustandsraumdarstellung in MATLAB. Berechnen Sie

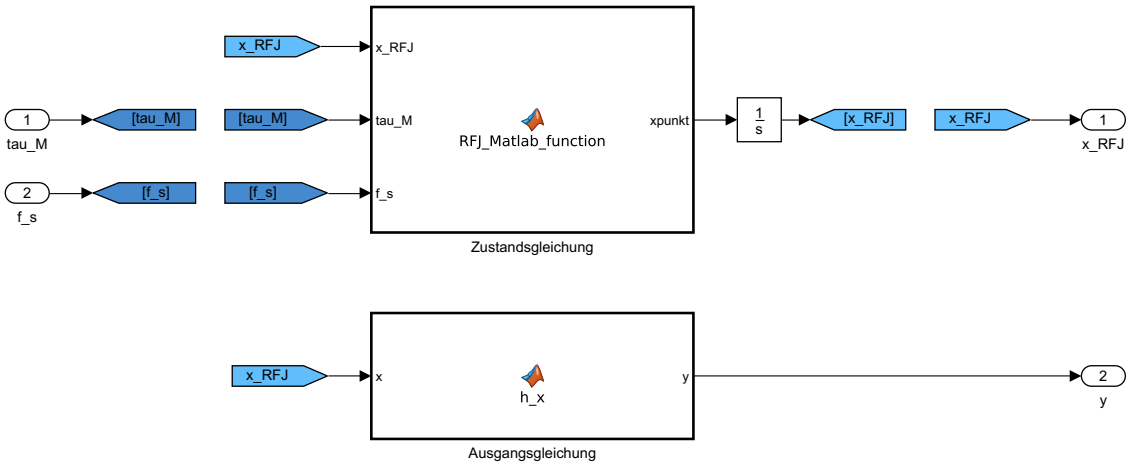

Abbildung 2.4: Struktur zur Implementierung des nichtlinearen Rotary Flexible Joint Modells in SIMULINK.

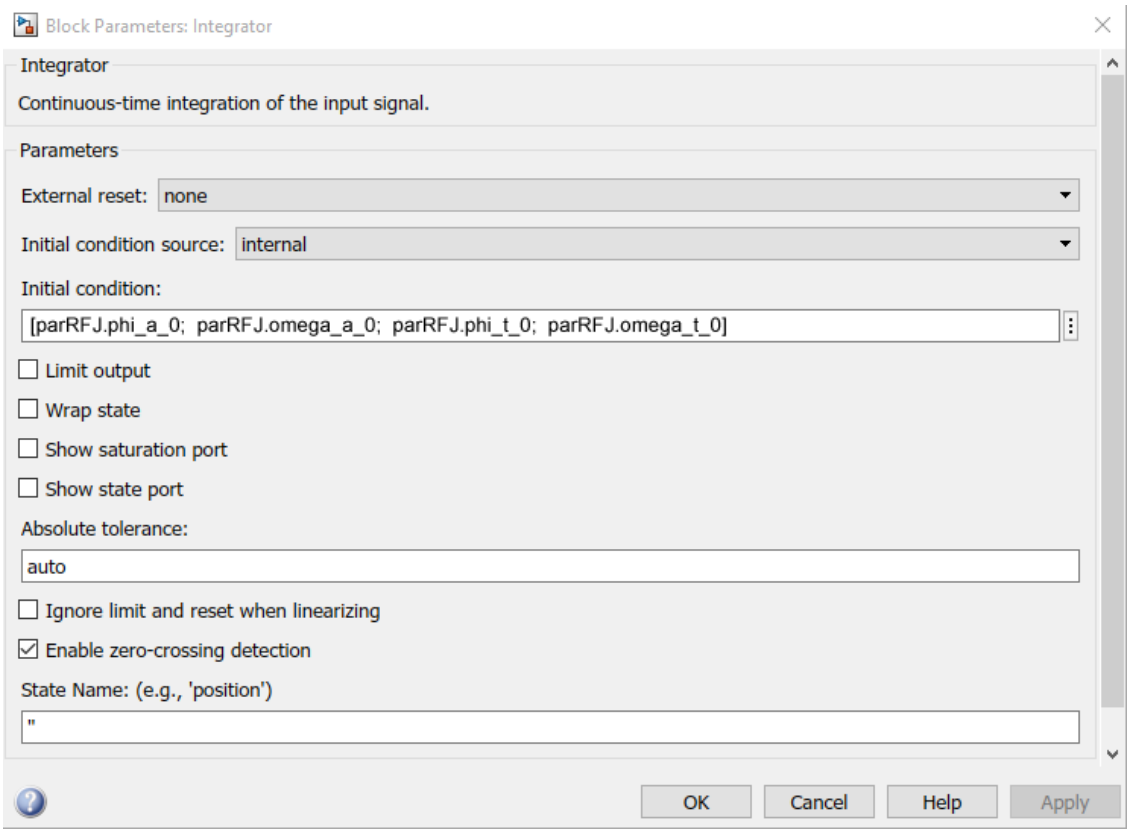

Abbildung 2.5: Berücksichtigung des Anfangszustandes $\mathbf{x}_0$ im Integrator-Block.

in MATLAB die zugehörige Führungsübertragungsfunktion $G_{u,y} = \frac{y(s)}{u(s)}$ vom Eingang

$u = \tau_{mot}$ zum Ausgang $y = \varphi_a$ und implementieren Sie diese mittels eines LTI-Blocks in Simulink. Überprüfen Sie die Implementierung des linearisierten Modells durch Vergleich mit der bereitgestellten `p-function`. Vergleichen Sie anschließend die Sprungantwort des nichtlinearen und linearisierten Modells.

Am Labormodell des Rotary Flexible Joint wurde hohe Reibung in den Lagern des Synchronmotors beobachtet, welche hauptsächlich durch den nichtlinearen Coulomb'schen Reibungsanteil bedingt ist. Zur Regelung des Systems hat sich folgende Vorgangsweise als zweckdienlich erwiesen. Da der Servoumrichter zur Ansteuerung des Synchronmotors über einen Drehzahlregler verfügt, kann ein kaskadierter Regelkreis aufgebaut werden, wobei der unterlagerte Regler zur Regelung der Winkelgeschwindigkeit ω_t mit einer wesentlich größeren Bandbreite als der des mechanischen Systems entworfen wird. Damit kann ω_t als Stellgröße des überlagerten Regelkreises verwendet werden.

Aufgabe 2.13. In Aufgabe 1.11 wurde das vollständige Modell um einen Zustand reduziert und das reduzierte Modell linearisiert. Implementieren Sie das nichtlineare, reduzierte Modell analog zu Aufgabe 2.11 und das linearisierte Modell des Rotary Flexible Joint in Form seiner Zustandsraumdarstellung analog zu Aufgabe 2.12 in MATLAB. Berechnen Sie in MATLAB die zugehörige Führungsübertragungsfunktion $G_{u,y} = \frac{y(s)}{u(s)}$ vom Eingang $u = \omega_t$ zum Ausgang $y = \varphi_a$. Vergleichen Sie anschließend die reduzierten Modelle durch Simulation in SIMULINK mit dem vollständigen nichtlinearen Modell.

Hinweis: Verwenden Sie für den Modellvergleich den Zustand ω_t des vollständigen Modells als Eingang $u = \omega_t$ der reduzierten Modelle.

Hinweis:

- Führen Sie alle Berechnungen in MAPLE mit allgemeinen Parametern durch und setzen Sie die Parameter erst dann ein, wenn eine numerische Auswertung verlangt ist (z.B. bei der Berechnung der Eigenwerte).
- Verwenden Sie zur übersichtlicheren Darstellung der Ergebnisse den Ersatzparameter $I_{a,zz}$ für das Trägheitsmoment des Auslegers um die z-Achse.
- Implementieren Sie die Matrizen des linearisierten Modells für allgemeine Parameter in einer MATLAB m-Datei. Die zugehörigen numerischen Werte der Parameter können Sie in der gleichen Datei vorher definieren.

2.2.2 Frequenzkennlinienverfahren

Aufgabe 2.14 (Reglerentwurf). Entwerfen Sie mithilfe des Frequenzkennlinienverfahrens im q-Bereich einen zeitdiskreten Kompensationsregler für das reduzierte linearisierte Modell des Rotary Flexible Joint aus Aufgabe 2.13. Gehen Sie dabei von einer Abtastzeit $T_a = 1\,\text{ms}$ aus. Die Sprungantwort des geschlossenen Kreises soll

folgende Spezifikationen erfüllen:

bleibende Regelabweichung	$e_\infty\big\vert_{(r^k)=(1^k)} = 0$
Anstiegszeit	$t_r = 0.1\,\mathrm{s}$
Überschwingen	$\ddot{u} \leq 10\,\%$

Als Reglerstruktur wird

$$R^{\#}(q) = V\frac{1 + 2\xi(qT) + (qT)^2}{(q - q_r)^2} \tag{2.15}$$

gewählt, wobei $1 + 2\xi(qT) + (qT)^2$ den Kompensationsterm, q_r den gewünschten doppelten Realisierungspol und V die Verstärkung bezeichnen.

1. Überlegen Sie, warum die obige Reglerstruktur gewählt wird.
2. Bestimmen Sie die Parameter T und ξ des Kompensationsterms so, dass dessen Nullstellen dem konjugiert komplexen Polpaar der Streckenübertragungsfunktion entsprechen.
3. Berechnen Sie q_r und V mithilfe des Frequenzkennlinienverfahrens entsprechend der obigen Spezifikationen.
4. Beurteilen Sie die Stabilität des geschlossenen Regelkreises.

Hinweis: In MATLAB stehen folgende Befehle für die Transformationen zwischen s-, z- und q-Bereich zur Verfügung:

$G(s) \overset{T_s}{\to} G(z)$	MATLAB-Befehl:	`Gz=c2d(Gs,Ts,'zoh')`
$G(z) \to G^{\#}(q)$	MATLAB-Befehl:	`Gq=d2c(Gz,'tustin')`
$G^{\#}(q) \overset{T_s}{\to} G(z)$	MATLAB-Befehl:	`Gz=c2d(Gq,Ts,'tustin')`

Machen Sie sich außerdem nochmals mit den Befehlen `tf()`, `bode()` (mit und ohne Rückgabeparameter), `feedback()` und `minreal()` vertraut, bevor Sie mit dem FKL-Entwurf starten.

Aufgabe 2.15 (Verifikation). Implementieren und testen Sie den entworfenen zeitdiskreten Regler in der SIMULINK-Datei `RFJ_Regelung.slx`. Die Verläufe von Führungs- und Störgröße sind bereits vorgegeben. Ergänzen Sie das reduzierte linearisierte Modell und den zeitdiskreten Regler an den vorgesehenen Stellen mithilfe von LTI-Blöcken.

1. Testen Sie durch Simulation, ob der entworfene Regler die Anforderungen für das reduzierte linearisierte System tatsächlich erfüllt.
2. Für das vollständige nichtlineare System ist ein unterlagerter Drehzahlregler

implementiert. Somit können Sie Ihren Regler auch für dieses System testen. Welche Unterschiede zum linearen reduzierten System stellen Sie fest? Wodurch entstehen diese Unterschiede?

3. Welches Verhalten beobachten Sie bei sprung- bzw. rampenförmiger Änderung der Führungsgröße und Störgröße?

4. Die Führungsgröße wird über ein Sollwertfilter geglättet. Warum ist der Einsatz eines solchen Filters sinnvoll? (Anmerkung: Für den Motor gilt eine Stellgrößenbeschränkung von $|\omega_t| \leq 15\,\mathrm{rad/s}$)

Hinweis: Die Visualisierung des RFJ in der Datei `RFJ_Regelung.slx` benötigt das Simscape Multibody-Addon, welches nicht standardmäßig in MATLAB installiert ist. Für den Fall, dass Sie dieses Addon nicht installiert haben, oder die Simulationszeit verkürzen wollen, können Sie den entsprechende Block auch auskommentierten.

Aufgabe 2.16 (Störübertragungsfunktion). Bei einer sprungförmigen Störung wird der Ausleger trotz Regelung in Schwingung versetzt. Berechnen Sie die Störübertragungsfunktion $T_{d,y}(z) = \frac{\hat{y}(z)}{\hat{d}(z)}$ des geschlossenen Kreises von der Störung $d = f_s$ zum Ausgang $y = \varphi_a$ und erklären Sie dieses Verhalten anhand des entsprechenden Bode-Diagramms.

2.3 Literatur

[1] A. Kugi, *Skriptum zur VU Automatisierung (WS 2018/2019)*, Institut für Automatisierungs- und Regelungstechnik, TU Wien, 2018.

[2] A. Angermann, M. Beuschel, M. Rau und U. Wohlfarth, *Matlab - Simulink - Stateflow, Grundlagen, Toolboxen, Beispiele*, 9. Aufl. Berlin: De Gruyter, 2017.

[3] H. Schwarz, *Numerische Mathematik*. Stuttgart: B.G. Teubner, 1997.