2 Systemanalyse mit Matlab/Simulink

Ziel dieser Übung ist es, das Computernumerikprogramm MATLAB und die zugehörige Simulationsumgebung SIMULINK zur Systemanalyse sowie zur Simulation dynamischer Systeme einzusetzen. Alle Aufgabenstellungen dieser Übungseinheit sind mit diesem Softwarepaket zu lösen. Im Computerlabor des Instituts steht MATLAB/SIMULINK in der Version R2016b zur Verfügung.

Studieren Sie als Übungsvorbereitung zumindest folgende Skripten:

- Skriptum zur VU Automatisierung (WS 2016/17) [2.1]
  - Kapitel 3, vollständig
  - Kapitel 6, bis Kapitel 6.4
- Skriptum zur VU Fachvertiefung Automatisierungs- und Regelungstechnik (WS 2016/17) [2.2]
  - Kapitel 3, vollständig
  - Abschnitt 4.7

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

- Christoph Fröhlich <froehlich@acin.tuwien.ac.at>
- Christian Krämer <kraemer@acin.tuwien.ac.at>.

2.1 Matlab

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

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**
  Im Editor können Funktionen, Skripte oder Programm-Code (z. B. C-Code) erstellt und editiert werden. Es werden die in Programmierumgebungen üblichen Möglichkeiten zum schrittweisen Ausführen der Befehlssequenzen, zum Debuggen, etc. zur Verfügung gestellt. Skripte werden auch M-files genannt; sie besitzen die Dateiendung `*.m` und werden zur Laufzeit von einem Interpreter abgearbeitet. Skripte enthalten MATLAB-Befehlssequenzen, wie sie prinzipiell auch direkt im Command Window eingegeben werden können. Funktionen besitzen ebenfalls die Dateiendung `*.m`. Sie erhalten meist Übergabeparameter und geben Rückgabewerte zurück.

- **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.

Als Beispiel für M-files können Sie die folgenden Dateien aus dem zip-Archiv `U2.zip` von der Homepage der Lehrveranstaltung (`http://www.acin.tuwien.ac.at/?id=61`) herunterladen.

- `cds_matlab_intro_part1.m`: Grundlegende Befehle.
- `cds_matlab_intro_part2.m`: Grafische Darstellung von Ergebnissen.
- `cds_matlab_intro_part3.m`: Beispiele zur Control System Toolbox.
- `mittelwert.m`: Beispiel einer Funktion (zur Berechnung des Mittelwertes zweier Zahlen).

2.1.1 Grundlegende Befehle

Aufgabe 2.1. Öffnen Sie die Datei `cds_matlab_intro_part1.m` im MATLAB-Editor und arbeiten Sie alle Befehle schrittweise durch. 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 Gleichungssystem

$$\begin{aligned} x_1 + 2x_2 + 4x_3 &= 2 \\ 2x_1 + 2x_2 + x_3 &= 3 \\ 3x_1 + 2x_2 &= 7. \end{aligned} \tag{2.1}$$

Schreiben Sie dieses Gleichungssystem in Matrixdarstellung an und bestimmen Sie den Lösungsvektor $\mathbf{x} = [x_1, x_2, x_3]^T$. 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. Öffnen Sie die Datei `cds_matlab_intro_part2.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.

Aufgabe 2.4. 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 eine geeignete Schrittweite Δx so, dass das Abtasttheorem erfüllt ist.

Aufgabe 2.5. Gegeben ist das Polynom

$$p_1(s) = s^3 + 3s^2 + 5s + 9, \tag{2.3}$$

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 $\mathbf{A}$ eine Hurwitzmatrix?

7. Zeigen Sie, dass die Matrix $\mathbf{A}$ ihr charakteristisches Polynom $p_2(s)$ erfüllt, also dem Satz von Cayley-Hamilton genügt (siehe Skriptum zur VU Automatisierung (WS 2016/17) [2.1] Satz 8.1). Sie können dazu den Befehl `polyvalm()` verwenden.

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.

2.1.2 Control System Toolbox

Toolboxen sind Sammlungen von Funktionen, meist in Form von M-files, die den Funktionsumfang des Basisprogramms erweitern. Nach der erstmaligen Installation werden Toolboxen automatisch beim Programmstart von MATLAB geladen. Eine Übersicht über die installierten Toolboxen erhält man mit dem Kommandozeilenbefehl `ver`.

Die Toolbox 'Control System' ist häufig bei regelungstechnischen Aufgabenstellungen nützlich. Sie unterstützt bei der Analyse und dem Reglerentwurf von linearen dynamischen Systemen.

Aufgabe 2.6. 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.

Zur Bestimmung der numerischen Lösung einer gewöhnlichen Differentialgleichung in Zustandsraumdarstellung

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}), \quad \mathbf{x}(0) = \mathbf{x}_0 \tag{2.5}$$

werden in MATLAB verschiedene Integrationsalgorithmen zur Verfügung gestellt.

Aufgabe 2.7.

1. Schreiben Sie ein M-file, in welchem die gewöhnliche Differentialgleichung

$$\dot{x}_1 = x_2, \qquad x_1(0) = 0, \tag{2.6a}$$
$$\dot{x}_2 = -x_1 - x_2 + u, \quad x_2(0) = 0 \tag{2.6b}$$

   mit Hilfe der MATLAB Funktion `ode45` gelöst wird. Wählen Sie hierzu den Simulationszeit $t_{sim} = 10\,\mathrm{s}$ und einen Einheitsprung $u = \sigma(t)$ als Eingangssignal. Die Funktion `ode45` arbeitet mit einer Schrittweitensteuerung, sodass das Eingangssignal zu den entsprechend richtigen Gitterpunkten t_k interpoliert werden muss. Hierzu kann der MATLAB Befehl `interp1` verwendet werden. Dieser interpoliert das Eingangssignal $u(t)$ auf dem Zeitgitter t am Gitterpunkt t_k und gibt den interpolierten Wert u_k zurück. Speichern Sie die Daten des Eingangssignals in der Struktur `input`.

2. Schreiben Sie ein weiteres M-file, in welchem das Differentialgleichungssystem (2.6) mit Hilfe des expliziten Euler-Verfahrens

$$\mathbf{x}_{k+1} = \mathbf{x}_k + T_a\,\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k), \tag{2.7}$$

   mit $\mathbf{x}_k = \mathbf{x}(kT_a)$, $k = 0, 1, \ldots$ und der Schrittweite $T_a = 0.5\,\mathrm{s}$ gelöst wird. Wählen Sie hierzu den Simulationszeitraum $t_{sim} = 10\,\mathrm{s}$, einen Einheitssprung $u = \sigma(t)$ als Eingangssignal. Geben Sie die Stabilitätsfunktion $R(\kappa)$ des expliziten Euler-Verfahrens (2.7) an und berechnen Sie die maximale Schrittweite $T_{a,max}$, für die das explizite Euler-Verfahren (2.7) das Differentialgleichungssystem (2.6) numerisch stabil löst. Verifizieren Sie Ihr Ergebnis simulativ.

3. Berechnen Sie unter Verwendung des Befehl `step()` der Control System Toolbox die Sprungantworten des kontinuierlichen Systems (2.6) und des zugehörigen Abtastsystems. Verwenden Sie zur Bestimmung des Abtastsystems den Befehl `c2d()` (Halteglied nullter Ordnung) mit einer Abtastzeit von $T_a = 0.5\,\mathrm{s}$. Vergleichen Sie Ihre numerischen Ergebnisse.

2.1.3 Simulink

SIMULINK ist eine Erweiterung von MATLAB zur Simulation und Analyse dynamischer Systeme. SIMULINK-Modelle besitzen die Dateiendung `*.slx`. Die grafische Bedienoberfläche erlaubt die Erstellung von Blockschaltbildern der untersuchten Modelle. Einerseits stellt SIMULINK eine Bibliothek mit vorgefertigten Funktionsblöcken zur Verfügung, andererseits können benutzerdefinierte Blöcke erstellt werden. Eine flexible Möglichkeit dafür sind so genannte S-function-Blöcke. S-functions können z. B. als MATLAB M-file oder in C programmiert werden. Sie erlauben die Implementierung dynamischer Modelle in einem Block.

Mit dem Befehl `simulink` wird der 'Simulink Library Browser' geöffnet. Er enthält die Funktionsblöcke, welche per Drag & Drop in das SIMULINK-Modell gezogen werden können. Die Blöcke sind mittels Signalflußleitungen zu verbinden. Besonders häufig benötigte Blöcke sind in der Gliederung des Simulink Library Browsers z. B. in den folgenden Gruppen zu finden.

- `Continuous`: Blöcke zur Simulation zeitkontinuierlicher Systeme, unter anderem der zeitkontinuierliche Integrator `1/s`.
- `Math Operations`: Einige mathematische Operationen, z. B. Addieren, Multiplizieren, Quadrieren.
- `Sinks`: Blöcke, die nur einen Eingang (mehrere Eingänge), aber keinen Ausgang besitzen. Beispielsweise werden `Scopes` zur grafischen Darstellung von Signalen genützt. Signale können an beliebiger Stelle direkt von Signalflußleitungen abgegriffen werden. Der Block `To Workspace` erlaubt den Export von Simulationsergebnissen in den MATLAB-Workspace, wo sie dann als Variable zur weiteren Auswertung zur Verfügung stehen.
- `Sources`: Blöcke, die nur einen Ausgang (mehrere Ausgänge), aber keinen Eingang besitzen, wie etwa Signalgeneratoren.

Zur effizienten Arbeit mit SIMULINK wird folgendes Vorgehen empfohlen:

1. Parameterwerte werden nicht direkt in SIMULINK eingetragen, sondern zusammengefasst in einem M-file definiert, welches vor der Simulation ausgeführt wird. Damit können die Parameter einfach und an zentraler Stelle geändert werden. Zum Update der Parameter muss das M-file 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. Für dynamische Systeme eignet sich der Block `level 2 Matlab S-Function`.

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.

Simulationseinstellungen können im Menü *Simulation* → *Model Configuration Parameters* vorgenommen werden. Besonders wesentlich ist dabei die Wahl der Simulationsdauer und des Integrationsalgorithmus. 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 Löser 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, 2.4], verwiesen.

Die Simulation kann durch Anklicken des Startknopfes ▶ oder durch Drücken der Tasten Strg und T gestartet werden. Um eine Simulation alternativ aus dem Eingabefenster oder einem M-file zu starten, kann der Befehl `sim()` verwendet werden.

Die Implementierung eines dynamischen Systems, für das ein Modell in Form einer (expliziten) Differentialgleichung existiert, kann entweder als Blockschaltbild oder als S-function erfolgen. Beides wird im Folgenden kurz erläutert.

Implementierung von dynamischen Systemen als Blockschaltbild

Aufgabe 2.8. Laden Sie die Dateien `set_params_simulink_testfile.m` und `simulink_testfile.slx` aus dem zip-Archiv `U2.zip` von der Homepage der Lehrveranstaltung (`http://www.acin.tuwien.ac.at/?id=61`) herunter. Führen Sie die Parameterdatei `set_params_simulink_testfile.m` aus und starten Sie die Simulation des Modells `simulink_testfile.slx`. Versuchen Sie anhand der nachfolgenden Beschreibung die Funktion aller Blöcke zu verstehen und machen Sie gegebenenfalls von der Hilfefunktion Gebrauch.

Lösung von Aufgabe 2.8. Das Modell enthält die Implementierung eines einfachen PT_2-Gliedes in Form von Blockschaltbildern. Die Übertragungsfunktion eines PT_2-Gliedes lautet bekanntlich

$$G(s) = \frac{V}{1 + 2\xi T s + (sT)^2}. \tag{2.8}$$

Um sie in Form eines Blockschaltbildes mit Integratoren (`1/s`) darzustellen, kann

zunächst die Beschreibung des Systems in Zustandsraumdarstellung

$$\dot{x}_1(t) = x_2(t) \tag{2.9a}$$

$$\dot{x}_2(t) = -\frac{1}{T^2}x_1(t) - \frac{2\xi}{T}x_2(t) + \frac{V}{T^2}u(t) \tag{2.9b}$$

mit den Anfangszuständen

$$x_1(0) = 0, \qquad x_2(0) = 0, \tag{2.10}$$

dem Ausgang $y = x_1$ und dem Eingang u oder in integrierter Form

$$x_1(t) = \int_0^t x_2(\tau)\mathrm{d}\tau \tag{2.11a}$$

$$x_2(t) = \int_0^t \left(-\frac{1}{T^2}x_1(\tau) - \frac{2\xi}{T}x_2(\tau) + \frac{V}{T^2}u(\tau)\right)\mathrm{d}\tau \tag{2.11b}$$

angeschrieben werden. Aus der Integraldarstellung (2.11) kann direkt eine mögliche Implementierung als Blockschaltbild in Simulink abgelesen werden. Dies ist in Abbildung 2.1 gezeigt. Anfangszustände ungleich Null können im Bedarfsfall als Parameter dem Block `1/s` übergeben werden.

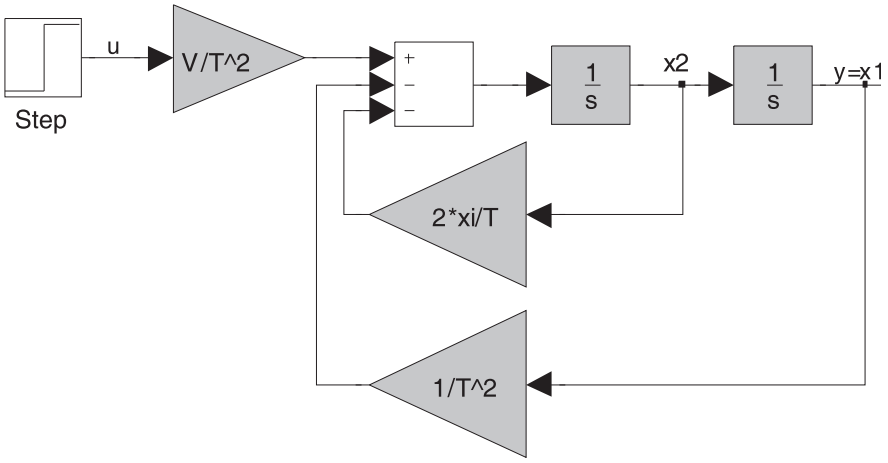

Abbildung 2.1: Mögliche Implementierung des PT_2-Gliedes in Simulink.

Für lineare zeitinvariante Systeme ist eine einfachere Implementierung mit Hilfe der Control System Toolbox möglich. Dabei kann ein LTI-System direkt in Form einer Übertragungsfunktion (z. B. (2.8)) oder einer Zustandsraumdarstellung (z. B. (2.9)) an den Block '`LTI System`' aus der Gruppe `Control System Toolbox` übergeben werden.

Aufgabe 2.9. Ermitteln Sie die numerische Lösung $x(t)$ der nichtlinearen Differentialgleichung

$$\dddot{x} + \cos(x)^2\ddot{x} + \dot{x} + e^{-x}u = 0 \tag{2.12}$$

wobei u den Eingang darstellt und für die Anfangsbedingungen $\ddot{x}(0) = 0$, $\dot{x}(0) = 0$ und $x(0) = 0$ gelten soll. Stellen Sie dazu die Differentialgleichung in Zustandsraumdarstellung dar und erstellen Sie das entsprechende Blockschaltbild in einem SIMULINK-Modell. Testen Sie die Simulation für $u(t) = \sin(t)$.

Implementierung von dynamischen Systemen als S-function

Sollen umfangreichere Systeme simuliert werden, geht die Übersichtlichkeit der in Form von Blockschaltbildern implementierten Modelle relativ rasch verloren. Hier kann die Verwendung von S-functions Abhilfe schaffen, wobei im Rahmen dieser Lehrveranstaltung ausschließlich so genannte *Level 2 Matlab S-functions* zur Anwendung kommen.

Das in Zustandsraumdarstellung gegebene System

$$\dot{\mathbf{x}} = \begin{bmatrix}\dot{x}_1\\ \dot{x}_2\end{bmatrix} = \begin{bmatrix}-x_2 + \sin(u_1)\\ a\,x_1 + b\,u_2\end{bmatrix} \tag{2.13a}$$

$$\mathbf{y} = \begin{bmatrix}x_1\\ x_2\\ e^{x_1}\cosh(x_2)\end{bmatrix} \tag{2.13b}$$

mit dem Eingangsvektor $\mathbf{u} = [u_1\ u_2]^{\mathrm{T}}$, den Parametern a und b, den Zuständen $\mathbf{x}$ und dem Ausgangsvektor $\mathbf{y}$ wurde in einer S-function implementiert.

Aufgabe 2.10. Laden Sie die S-function-Datei `test_sfunc_m.m` gemeinsam mit dem Simulationsmodell `test_sfunc.slx` aus dem zip-Archiv `U2.zip` von der Homepage der Lehrveranstaltung (`http://www.acin.tuwien.ac.at/?id=61`) herunter. Arbeiten Sie die S-function durch und führen Sie die Simulation aus. Versuchen Sie den Aufbau der S-function anhand der nachfolgenden Beschreibung zu verstehen und machen Sie gegebenenfalls von der Hilfefunktion Gebrauch.

Das zentrale Element einer *Level 2 Matlab S-function* ist ein run-time Objekt - eine Instanz der Klasse `Simulink.MSFcnRunTimeBlock`. Das Objekt wird üblicherweise als `block` bezeichnet. Die Attribute des Objekts beinhalten für die Simulation wichtige Eigenschaften und Funktionen. Auf sie kann mit Hilfe des Punkt-Operators zugegriffen werden. Tabelle 2.1 fasst einige wichtige Attribute zusammen. Zusätzlich können einer S-function Parameterwerte, z. B. physikalische Parameter des Systems, übergeben werden. Damit ist die Änderung von Parametern direkt in SIMULINK bzw. von MATLAB aus möglich, ohne die S-function selbst zu verändern.

Typischerweise enthält eine S-function für dynamische Systeme die folgenden Funktionen (siehe auch das Beispiel `test_sfunc_m.m`).

Objektname.Attributname	Erklärung
`block.InputPort`	Eingänge $\mathbf{u}$
`block.ContStates`	Zustände $\mathbf{x}$
`block.Derivatives`	Zeitableitung der Zustände $\dot{\mathbf{x}}$
`block.OutputPort`	Ausgänge $\mathbf{y}$
`block.DialogPrm`	An die S-function übergebene Parameter des Systems

Tabelle 2.1: Wichtige Attribute des run-time Objekts `block`.

1. `function setup(block)`
   In dieser Funktion wird das run-time Objekt `block` initialisiert. Dabei wird z. B. die Länge der Eingangs-, Zustands-, Parameter- und Ausgangsvektoren festgelegt. Es ist möglich, Eingänge und Ausgänge zu so genannten Ports zu gruppieren. In der Beispieldatei `test_sfunc_m.m` wurden zwei Ausgangsports festgelegt, wobei Port 1 die Ausgangssignale x_1 und x_2 zusammenfasst und Port 3 den dritten Ausgang enthält, welcher durch die Funktion $e^{x_1}\cosh(x_2)$ gegeben ist. Die beiden Eingänge werden jeweils über separate Ports an die S-function übergeben. Der Parametervektor ist kein herkömmlicher MATLAB-Vektor, da dessen Elemente unterschiedliche Datentypen aufweisen dürfen. In der Beispieldatei ist das dritte Element ein Vektor. Sollen sehr viele Parameter übergeben werden, ist die Verwendung von `Structures` zu empfehlen. Die verwendeten Feldnamen stehen dann auch in der S-function zur Verfügung, womit der Verwechslung von Parametern vorgebeugt werden kann.

2. `function InitConditions(block)`
   In dieser Funktion werden den Zuständen die Anfangswerte zugewiesen. Günstigerweise werden diese im Parametervektor an die S-function übergeben.

3. `function Output(block)`
   Berechnung der Ausgangsgleichung
   $$\mathbf{y} = \mathbf{g}(\mathbf{x}, \mathbf{u}). \tag{2.14}$$

4. `function Derivatives(block)`
   Diese Funktion kommt nur bei zeitkontinuierlichen dynamischen Systemen vor. Es werden die Ableitungen berechnet, d. h. die Differentialgleichung wird in der Form
   $$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}) \tag{2.15}$$
   eingegeben.

5. `function Update(block)`
   Diese Funktion kommt bei zeitdiskreten dynamischen Systemen vor. Es wird die Differenzengleichung in der Form
   $$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) \tag{2.16}$$
   implementiert.

Die obigen Funktionen werden zur Laufzeit der Simulation von SIMULINK aufgerufen — zum Teil auch mehrmals, wie beispielsweise die Funktionen `Output(block)`, `Derivatives(block)` oder `Update(block)`. Bei den angegebenen Funktionen in der Beispieldatei `test_sfunc_m.m` ist zu beachten, dass meist neue Variablen für die Attribute des Objektes `block` eingeführt wurden. Dies dient der besseren Lesbarkeit der Programme und kann den Schreibaufwand verringern.

Beachten Sie im Zusammenhang mit S-functions folgende Hinweise:

- Der Name des M-files, welches die S-function beinhaltet, muss mit dem Namen der S-function (in diesem Fall: `test_sfunc_m`) übereinstimmen.
- Das SIMULINK-Modell (`*.slx`) und die `S-function` (`*.m`) müssen verschiedene Namen haben, sich aber im selben Verzeichnis befinden.
- Verwenden Sie zum Einbinden von S-functions in SIMULINK-Modelle den Block `User-Defined Functions` → `level 2 Matlab S-Function`.
- S-function-Blöcke können maskiert werden, so dass man die Parameter direkt in einer Maske eingeben kann (siehe Beispiel `test_sfunc.slx`).

Aufgabe 2.11. Erstellen Sie für das System Gleichstrommaschine mit Propeller nach Abschnitt 1.4 ein SIMULINK-Modell, in welchem das *vollständige* nichtlineare Modell (inklusive Stromdynamik nach Aufgabe 1.5, Teilaufgaben 2 bis 4) mit Hilfe einer Level 2 Matlab S-function simuliert werden kann. Wählen Sie als Eingangsvektor der S-function $\mathbf{u} = [u_{GSM}, M_{ext}]^T$ und als Ausgangsvektoren $\mathbf{y}_1 = [i_{GSM}, \varphi_{GSMP}, \omega_{GSM}, \omega_P]^T$ und $\mathbf{y}_2 = [M_{GSM}, M_{Kopp}]^T$. Hierbei bezeichnet M_{GSM} das elektrische Moment der Gleichstrommaschine und M_{Kopp} das Kopplungsmoment zwischen Gleichstrommaschine und Propeller. Verwenden Sie als Anfangszustand die Ruhelage für $u_{GSM} = 5.6\,\mathrm{V}$ und $M_{ext} = 0\,\mathrm{Nm}$. Die Eingangsgrößen haben die Verläufe $u_{GSM}(t) = 5.6 - 1\sigma(t-2) + 2\sigma(t-5) - 2\sigma(t-8) + 4\sigma(t-13)$ und $M_{ext}(t) = 0.25\sigma(t-11)$ (u_{GSM} in V und M_{ext} in Nm).

Beachten Sie bei der Implementierung folgende Hinweise:

- Zur einfacheren Wiederverwendung Ihres Simulationsmodells in späteren Übungsaufgaben sind die oben angegebenen Reihenfolgen der Ein- und Ausgänge exakt einzuhalten.
- Alle Systemparameter und Anfangszustände sollen als Parameter von außen an die S-function übergeben werden. Definieren Sie diese Größen in einem M-file, das Sie jeweils vor dem Start der Simulation ausführen (ähnlich der Parameterdatei `set_params_simulink_testfile.m` in Aufgabe 2.8).
- Übernehmen Sie dabei die *analytischen* Ausdrücke der Ruhelagen und des linearisierten Modells aus der in Aufgabe 1.5, Teilaufgabe 4 erstellen MAPLE-Arbeitsblatt-Datei in besagtes M-file und berechnen Sie erst dort die numerischen

Werte. Damit ist es später einfach möglich, beliebige Ruhelagen zu untersuchen.

Kontrollhinweis: Zum Test und zum Abgleich ihres Modells steht im zip-Archiv `U2.zip` auf der Homepage der Lehrveranstaltung (`http://www.acin.tuwien.ac.at/?id=61`) die Datei `GSM_Student_S_m.p` zum Download zur Verfügung. Es handelt sich um eine chiffrierte S-function der Gleichstrommaschine. Ihre Einbindung erfolgt analog zur Level 2 Matlab S-function und ist im Simulationsmodell `Simulation_GSM_out.slx` gezeigt. Das Modell enthält bereits die oben genannten Verläufe der Eingangsgrößen. Um Namenskonflikte zu vermeiden, darf die von Ihnen erstellte S-function *nicht* den Namen `GSM_Student_S_m.m` tragen.

Aufgabe 2.12. Implementieren Sie das um die Stromdynamik reduzierte und um die Ruhelage $u_{GSM} = 5.6\,\mathrm{V}$ linearisierte Modell aus den Zusatzaufgaben der Übung 1 in Zustandsraumdarstellung mit Hilfe der Control System Toolbox in SIMULINK. Achten Sie auf eine korrekte Arbeitspunktaufschaltung. Verwenden Sie die Ruhelage als Anfangszustand und die gleichen Eingangsgrößen wie in Aufgabe 2.11. Vergleichen Sie die Ergebnisse mit jenen des vollständigen nichtlinearen Modells aus Aufgabe 2.11.

Implementierung von zeitdiskreten dynamischen Systemen als Matlab-function

Da Regelungs- und Beobachterstrategien meist in Digitalrechnern implementiert werden, sind diese als *zeitdiskrete* dynamische Systeme zu berücksichtigen. Dazu kann alternativ zu einer S-function, eine so genannte *Matlab-function* genutzt werden. Mit dieser ist es wie in einer S-function möglich komplexere Rechnungen durchzuführen, da auf fast alle Matlab Funktionen zurückgegriffen werden kann. Die Matlab-function hat außerdem den Vorteil, dass sie in M-Code programmiert werden kann und dieser vor dem Starten der Simulation in eine 'C-mex-function' kompiliert wird. Neben reduzierter Rechenzeit bietet das vor allem den Vorteil, dass Matlab-functions direkt in dem Rapid Prototyping System DSPACE verwendet werden können.

Das zeitkontinuierliche System (2.13) wurde in einer Matlab-function implementiert, wobei die Zeitableitung über das explizite Euler-Verfahren diskretisiert wurde.

Aufgabe 2.13. Laden Sie die Dateien `test_matlabfunc.slx`, `set_params_test_matlabfunc.m` und `PlotXPhasenraum.m` aus dem zip-Archiv `U2.zip` von der Homepage der Lehrveranstaltung (`http://www.acin.tuwien.ac.at/?id=61`) herunter. Arbeiten Sie die Matlab-function durch, führen Sie die Parameterdatei `set_params_test_matlabfunc.m` aus und starten Sie die Simulation. Versuchen Sie den Aufbau der Matlab-function anhand der nachfolgenden Beschreibung zu verstehen und machen Sie gegebenenfalls von der Hilfefunktion Gebrauch.

Durch Doppelklick auf die Matlab-function, wechselt MATLAB in den Editor. Zur Implementierung eines zeitdiskreten dynamischen Systems müssen folgende Punkte eingestellt werden:

1. Mit dem Button `Edit Data` wechselt MATLAB in den 'Ports and Data Manager', in dem im ersten Schritt die Ein- und Ausgänge, die Parameter sowie das Update Verhalten des Blocks eingestellt werden müssen. Da hier ein zeitdiskretes dynamisches System mit der fixen Abtastzeit T_a implementiert werden soll, muss bei 'Update method' über das Popup Menü 'Discrete' eingestellt werden und anschließend unter 'Sample Time' die Abtastzeit eingetragen werden. Auf der linken Seite des Menüs können Ein- und Ausgänge sowie Parameter hinzugefügt und umbenannt werden.
2. Wie bei normalen MATLAB Funktionen üblich sind innerhalb der Funktion definierte Variablen nur lokal bekannt und werden am Ende der Funktion wieder gelöscht. Soll der Wert einer Variablen auch beim nächsten Aufruf der Funktion noch bekannt sein, muss diese Variable auf `persistent` gesetzt und vor dem ersten Gebrauch initialisiert werden.
3. Anschließend kann wie in MATLAB üblich der restliche Funktionsteil geschrieben werden (vgl.: `mittelwert.m`).
4. In einer Matlab-function sind nicht alle in MATLAB definierte Funktionen direkt aufrufbar. Sollen eigene Funktionen verwendet werden, muss der Compiler diese mit kompilieren. Dies kann dadurch erreicht werden, indem man die Zeile `%#codegen` direkt unter den Funktionsheader der eigenen Funktion eingibt (vgl.: `PlotXPhasenraum.m`). Sollen weitere, in MATLAB schon vorkompilierte und standardmäßig nicht eingebundene Funktionen dazu gelinkt werden, so kann man diese durch die Zeile `coder.extrinsic{functionname}` hinzufügen.

Aufgabe 2.14.

1. Implementieren Sie das mit dem Befehl `c2d()` diskretisierte System (Abtastzeit $T_a = 0.1s$) aus Aufgabe 2.7 in Form einer Matlab-function.
2. Implementieren Sie das System außerdem in einem LTI-Block und vergleichen Sie die Ergebnisse für verschiedene Eingangsfunktionen.

2.2 Literatur

[2.1] A. Kugi, *Skriptum zur VU Automatisierung (WS 2016/2017)*, Institut für Automatisierungs- und Regelungstechnik, TU Wien, 2016. Adresse: http://www.acin.tuwien.ac.at/?id=42.

[2.2] A. Kugi und W. Kemmetmüller, *Skriptum zur VU Fachvertiefung Automatisierungs- und Regelungstechnik (WS 2016 /2017))*, Institut für Automatisierungs- und Regelungstechnik, TU Wien, 2016. Adresse: http://www.acin.tuwien.ac.at/?id=46.

[2.3] A. Angermann, M. Beuschel, M. Rau und U. Wohlfarth, *Matlab - Simulink - Stateflow, Grundlagen, Toolboxen, Beispiele*. München: Oldenbourg Verlag, 2005.

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