RWTH AACHEN
RHEINISCH-WESTFÄLISCHE TECHNISCHE HOCHSCHULE AACHEN

IRT
INSTITUT FÜR REGELUNGSTECHNIK
UNIV.-PROF. DR.-ING. DIRK ABEL

Kurzeinführung in

MATLAB/SIMULINK/STATEFLOW

Version 2.6 vom 5.02.2007

Inhaltsverzeichnis

1 Einführung in Matlab

MATLAB ist eins der verbreitesten Programme, insbesondere in der Regelungstechnik, zum wissenschaftlichen, numerischen Rechnen. Neben der Berechnung an sich bietet MATLAB auch Möglichkeiten zur graphischen Darstellung und zum Programmieren eigener Skripte. Es handelt sich um ein interaktives System mit Matrizen als Basis-Datenelement. MATLAB steht daher für Matrizen Laboratorium. SIMULINK ist eine auf MATLAB aufgesetzte graphische Benutzeroberfläche, mit der komplexe Systeme modelliert und simuliert werden können. Die notwendigen Matrixberechnungen werden von MATLAB automatisch durchgeführt. Hersteller dieser beiden Produkte ist die Firma The Math Works Inc. in den USA.

Das Ziel der Übung ist eine erste Einarbeitung in MATLAB/SIMULINK insbesondere auf dem Gebiet der Regelungstechnik. Bei dieser Übungsanleitung steht weniger die Vollständigkeit der Darstellung als vielmehr die Anleitung zum selbständigen weiteren Arbeiten im Vordergrund. Kleinere Übungen zum besseren Verständnis sind innerhalb der Kapitel integriert. Die mit dem Handsymbol „☞“ markierten Aufgaben am Ende jedes Kapitels dienen der Überprüfung der erworbenen Kenntnisse. Darüber hinaus werden an verschiedenen Stellen Vorschläge zum selbständigen Experimentieren gemacht.

Die Kapitel eins bis vier orientieren sich sehr stark an dem Buch: „Matlab, Simulink, Stateflow“ von M. Rivoire und J.-L. Ferrier. Den Autoren sei ein großer Dank ausgesprochen!

Bitte beachten Sie: Diese Anleitung ist auf dem Stand von MATLAB **Version 7.3 (R16)**. Sollten Sie mit einer neueren Version arbeiten, können sich insbesondere Menüstrukturen und Dialogboxinhalte verändert haben. Sollten Sie die beschriebenen Befehle oder Funktionen nach etwas Suchen nicht finden können, wenden Sie sich bitte an den Betreuer.

1.1 Erste Schritte und Bedienungshilfen...

Befehlsfenster

Nach dem Starten von MATLAB erscheint die MATLAB-Oberfläche (siehe Abb. 1.1). Sie besteht aus mehreren Fenstern, wobei das Befehlsfenster bzw. **Matlab Command Window** mit dem MATLAB-Prompt ≫ den größten Platz einnimt. Das Befehlsfenster dient zur Eingabe von Variablen und Ausführen von Befehlen, Funktionen und Skripte. Jede Befehlszeile wird mit einem Return ↵ abgeschlossen und der Ausdruck direkt nach der Eingabe ausgewertet. In diesen Unterlagen enthaltene Beispiele beginnen mit dem oben angegebenen Prompt. Kommentare, denen ein Prozentzeichen % vorangestellt ist, brauchen nicht eingegeben zu werden.

Versuchen Sie, die folgenden Beispiele nachzuvollziehen:

```
≫ 5* 7          % Zuweisung der Standard-Variablen ans
≫ a=3* pi       % Zuweisung der Variablen a
≫ b = a/5
≫ c = a* b;
≫ b = 1:3:20
≫ plot(b)
≫ help cos
≫ b = 2cos(b)
```

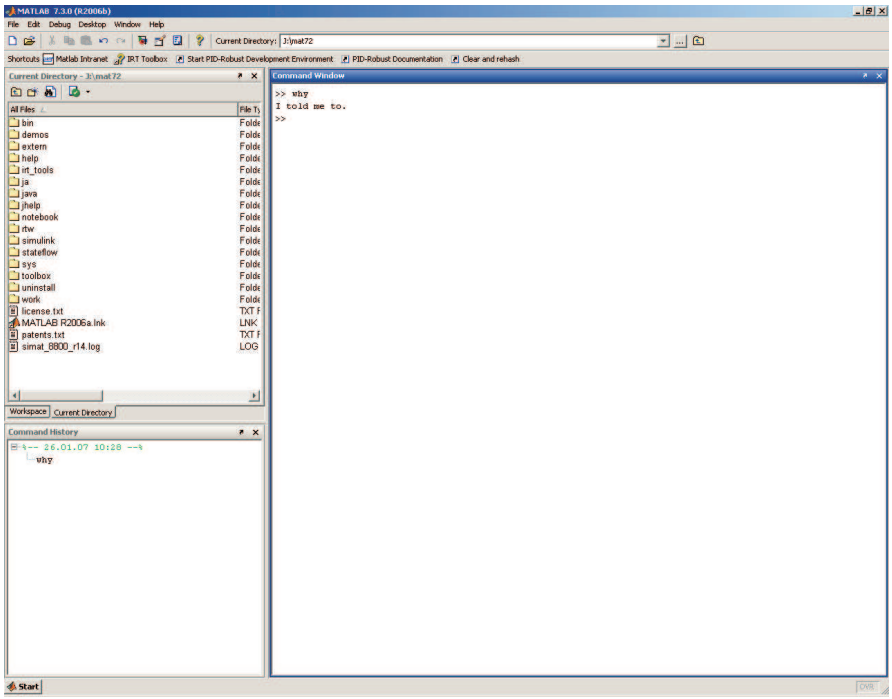


Abbildung 1.1: MATLAB Oberfläche

Die letzte Zeile verursacht eine der häufigsten Fehlermeldungen: **missing operator, comma, or semi-colon**. In diesem Fall fehlt der $*$ zur Multiplikation zwischen 2 und dem cosinus. Der korrekte Befehl lautet:

$\gg b = 2 * cos(b)$

Die Ausgabe auf dem Bildschirm kann mit einem Semikolon am Ende der Zeile unterdrückt werden. Testen Sie dieses, indem Sie den Befehl mit einem Semikolon wiederholen.

Anstatt den Befehl neu wiedereinzugeben, kann auch die Taste ↑ (Pfeil nach oben) gedrückt werden, mit der eingegebene Befehle wiederholt werden. Diese Befehle können vor Eingabe des Return ↵ beliebig verändert werden. Noch schneller geht es, wenn der oder die ersten Buchstaben des Befehls bekannt sind. Gibt man diese ein und drückt dann die Taste ↑, werden nur diejenigen Befehle mit demselben Beginn wiederholt.

Welcher Befehl erscheint somit bei Eingabe von **p** ↑?

Symbolleiste des Befehlsfensters

Die Symbolleiste (vgl. Abb. 1.1) ermöglicht den schnellen Zugriff auf häufig benötigte Funktionen. Die Bedeutung der einzelnen Symbole werden von links nach rechts erklärt:

- Starten des MATLAB-Editors/Debugger mit einem neuen Fenster
  Dieser Vorgang ist analog zu der Menü-Auswahl **File:New:M.File**. Der Editor und das Erstellen bzw. Verändern eigener Skripte wird später erläutert.
- Öffnen eines existierenden MATLAB-Dokuments im MATLAB-Editor/Debugger

Es wird eine Auswahl mit der Endung **.m** für Matlab-Skripte, **.mdl** für SIMULINK-Modelle oder **.fig** für Matlab-Bilder angezeigt. Grundsätzlich kann aber auch jede andere ASCII-Datei im Editor bearbeitet werden.

- Ausschneiden, Kopieren und Einfügen
- Rückgängig und Wiederherstellen
- Starten von SIMULINK
- Starten von GUIDE
  Ein offener GUI Editor
- Starten von Profiler
  zur Optimierung von M-Files
- Öffnen des Hilfefensters
  Das Hilfefenster enthält den gleichen Hilfetext zu einer beliebigen Funktion *fname* wie bei Eingabe von **help *fname*** im Befehlsfenster. Der Text wird aber nicht im Befehls- sondern in einem gesonderten Fenster, welches auch ein einfacheres Navigieren zwischen den Befehlen erlaubt, angezeigt.
- Aktuelles Verzeichnis (Current Directory)
  In diesem Verzeichnis wird als erstes nach benötigten Dateien gesucht und werden zu speichernde Dateien abgelegt. Ist eine Suche im aktuellen Verzeichnis nicht erfolgreich, wird sie dem Suchpfad folgend fortgesetzt. Der Suchpfad kann mit dem Menüpunkt **File - Set Path** festgelegt werden.
  In diesem Zusammenhang gibt es noch ein paar nützliche Befehle im Befehlsfenster:

```
» cd        % Zeigt den aktuellen Pfad
» cd ..     % Wechsel in das übergeordnete Verzeichnis
» cd work   % Wechsel in das untergeordnete Verzeichnis work
» dir       % Listet alle Dateien im aktuellen Verzeichnis auf
» what      % Listet alle MATLAB/SIMULINK-Dateien im Verzeichnis auf
```

Weitere Fenster

Alle verwendeten Variablen werden automatisch im sogenannten **Workspace** gespeichert. Im gleichnamigen Fenster werden Größe und Typ der verwendeten Variablen angezeigt. Durch Doppelklick auf die Variablen lassen sich deren Inhalt und Eigenschaften anzeigen und verändern.

Das Fenster **Current Directory** zeigt den Inhalt des aktuellen Verzeichnisses in der üblichen Baumstruktur an. Darin sind auch Dateioperationen wie Öffnen, Löschen, Umbenennen und Kopieren in gewohnter Weise möglich.

Im Fenster **Command History** wird eine Liste der zuletzt eingegebenen Befehle angezeigt. Durch Doppelklick auf einen Befehl wird dieser erneut ausgeführt.

Hilfe

Meistens benötigt man die direkte Hilfe zu einer Funktion, insbesondere zu ihrer Syntax. Dazu gibt man den Befehl **help** (oder **doc** für mehr Details) gefolgt vom Funktionsnamen ein:

```
» help cos
» help linspace
» help plot
» doc plot
» lookfor plot
```

Der Befehl **lookfor** durchsucht in den Dateien im Suchpfad alle ersten Zeilen des Hilfetexts nach dem angegebenen Wort.

Über die Eingabe von Befehlen lassen sich weitere Hilfemöglichkeiten aufrufen:

- Help Window
  Eigenes Hilfefenster zum Anzeigen des Hilfetexts einer Funktion (s. Symbolleiste). Dazu kann durch Doppelklicken auf einen Eintrag in der Gliederung **Help Topics** ein Bereich durchsucht werden. Das Hilfefenster erscheint ebenfalls durch Eingabe des Befehls **helpwin** im Befehlsfenster.

- Help Desk
  Das Help Desk enthält in Form von .html oder .pdf Dateien ausführlichere Informationen zu den einzelnen Funktionen. Das Help Desk erscheint ebenfalls durch Eingabe des Befehls **helpdesk** im Befehlsfenster.

- Examples und Demos
  Es erscheint das Demo Fenster mit einer Liste von mitgelieferten Beispielen und Demonstrationen. Das Demo Fenster erscheint ebenfalls durch Eingabe des Befehls **demo** im Befehlsfenster.

Aufgaben und Übungen

1.1 ☞ Ändern Sie den aktuellen Pfad auf das von Ihrem Betreuer angegebene Arbeitsverzeichnis und fügen sie das neue Arbeitsverzeichnis dem Suchpfad hinzu.

1.2 ☞ Erklären Sie kurz die Bedeutung und Syntax der Befehle **exp**, **grid**, **figure** und **hold**.

1.3 ☞ Ermitteln Sie die Bedeutung der Befehle **which**, **pause** und **disp**.

1.4 ☞ Erzeugen Sie mit Hilfe des Befehls **linspace** eine Zahlenfolge mit 22 Werten zwischen 1.5 und 9.

1.5 ☞ Öffnen Sie eine neue Datei im Editor und tragen Sie die Befehle

```
% Das ist der erste Versuch!
cd
disp('Guten Tag!')
a=5*7
b=5*7;
```

ein. Speichern Sie die Datei in Ihrem Arbeitsverzeichnis unter dem Namen **versuch1.m**. Schließen Sie den Editor. Geben Sie im Befehlsfenster den neuen Befehl **versuch1** und **help versuch1** ein. Was macht der neue Befehl?

Wechseln Sie in das Fenster **Current Directory**. Öffnen Sie dort Ihre Datei versuch1.m durch Doppelklick. Drücken Sie im Editor die Taste **F5**. (Damit speichern Sie die Datei und bringen sie sofort zur Ausführung.)

1.2 Konstanten und Variablen

Konstanten

MATLAB wurde speziell zur Berechnung von Matrizen geschrieben. Eine Matrix ist für MATLAB ein Feld mit numerischen Einträgen. Wörter (Strings) sind Zeichen-Felder, bei denen die Feldelemente den ASCII-Code (ganzzahlige, positive Werte) der Buchstaben enthalten. MATLAB kennt zwar auch noch andere Wege, alphanumerische Daten zu speichern, jedoch bleiben wir vorerst bei der Vorstellung von numerischen Matrizen.

Konstanten (Skalare) sind intern 1×1-Matrizen. Vektoren sind $1 \times n$-Matrizen (Zeilenvektoren) oder $n \times 1$- Matrizen (Spaltenvektoren). Bei Matrix-Berechnungen (z. B. Multiplikation) muss immer genau der Überblick behalten werden, was man berechnen möchte und welche Art von Vektoren dazu verwendet werden. Sonst erhält man als Ergebnis nur Fehlermeldungen oder nicht beabsichtigte Ergebnisse, da etwas anderes berechnet worden ist!!!

Matrizen mit mehr als einem Element werden zeilenweise eingegeben und mit eckigen Klammern [ ] begrenzt. Innerhalb einer Matrix dienen Kommata oder Leerschritte zur Spaltentrennung und Semikoli zur Zeilentrennung:

```
» 5
» 3.14159
» 2.0e-10
» [1,2,3]                    % [ 1  2  3 ] Zeilenvektor
» [4;5;6]                    % [ 4 ; 5 ; 6 ] Spaltenvektor
» A = [1,2,3;4,5,6;7,8,0]    % [ 1 2 3 ; 4 5 6 ; 7 8 0 ] Matrix
» b = [1,2,3]
» c = 'test'
```

Vektoren können auch schneller in der folgenden Form eingegeben werden:

vektor = [*1.Element* : *Inkrement* : *letztes Element*];

Der Vektor enthält anschließend äquidistante Einträge. Das Inkrement kann auch negativ sein und im Falle, dass es 1 ist, weggelassen werden.

```
» [1 : 10]
» [1 : -1.5 : -8]
```

Desweiteren kann man mit den Funktionen **linspace** und **logspace** Vektoren erzeugen. linspace ist bereits aus dem vorigen Abschnitt bekannt. Wodurch unterscheidet sich logspace von linspace?

Variablen

Wie in jeder Programmiersprache können auch in MATLAB Variablen definiert werden. Die Zuweisung erfolgt mit dem Gleichheitszeichen =. Jede Zeichenfolge, die mit einem Buchstaben beginnt und keine Sonderzeichen enthält, darf als Variablenname verwendet werden. Groß- und Kleinschreibung wird unterschieden. MATLAB benötigt keine vorangehenden Deklarationen, d. h. eine Variable wird automatisch neu deklariert, wenn sie auf der linken Seite einer Zuweisung auftaucht. Auf der rechten Seite dürfen nur bereits bekannte Variablen oder Konstanten verwendet werden. Rekursive Zuweisungen sind möglich.

Wie bereits erwähnt, basieren die Variablen von MATLAB auf Matrizen. Diese können sowohl Zahlen als auch Buchstaben enthalten. Neben den Zahlen und Buchstaben gibt es noch sogenannte **Zellen** und **Strukturen**. Aus zeitlichen Gründen beschränkt sich diese Einführung aber hauptsächlich auf die wesentlichen numerischen Matrizen. Wer mehr wissen möchte, sei entweder auf den Hilfetext oder das Handbuch von MATLAB verwiesen. Es darf aber auch der Betreuer gefragt werden . . . .

Analog zum Workspace Browser wird mit den Befehlen **who** (Namen der verwendeten Variablen) und **whos** (Namen und Größe der verwendeten Variablen) der Inhalt des Workspace im Befehlsfenster angezeigt.

```
» b* A
» d = 2*A;
» d
» who
» whos
```

Außerdem gibt es noch einige reservierte, vorbesetzte Systemvariablen, die nicht verändert werden sollten:

- **ans** enthält das letzte Ergebnis eines Befehls, falls keine Zuweisung zu einer anderen Variable erfolgt ist.
- **pi** enthält die Zahl π.
- **eps** enthählt die Zahl $=2^{-52} = 2.22e - 16$.
- **realmin, realmax** sind der kleinste und größte reelle Wert, die in MATLAB verfügbar sind.
- **Inf** (infinit: unendlich) steht für $1/0$
- **NaN** (Not a Number) steht für $0/0$
- **i, j** enthalten beide die imaginäre Einheit $\sqrt{-1}$ zur Eingabe von komplexen Zahlen.

Falls aus Versehen eine Systemvariable *vname* mit einem anderen Wert überschrieben worden ist, so erhält man den ursprünglichen Wert mit dem Befehl **clear *vname*** wieder, der die entsprechende Variable aus dem Workspace löscht.

```
» pi = 2.3
» clear pi
» pi
» komplex = 3 + 2 * i
```

Möchte man auf einzelne Werte aus einer Matrix zugreifen, so werden die entsprechenden Zeilen- und Spaltennummern in runde Klammern gesetzt. Die Zeilen- und Spaltennummern können auch wieder Vektoren sein, um gleichzeitig auf mehrere Zeilen oder Spalten zugreifen zu können.

```
» A(1,3)
» A(1,:)        % ':' repräsentiert ganze Zeile
» A(:,2)        % ':' repräsentiert ganze Spalte
» A([1 3],1)
```

Logische Matrizen helfen, Werte, die bestimmten Bedingungen genügen, aus einer Matrix z. B. A zu extrahieren. Dazu muss die logische Matrix mit der gleichen Dimension wie A und der entsprechenden Bedingung, z. B. Werte zwischen 2 und 5, aufgebaut werden. Anschließend können die Werte extrahiert werden. Das folgende Beispiel verdeutlicht das Vorgehen:

```
» X=(A>= 2 & A <= 5)
» A(X)
```

Mit Hilfe einer leeren Matrix [ ] können wieder Zeilen oder Spalten einer Matrix gelöscht werden:

```
» A(:,2) =[ ]   % Die zweite Spalte von A wird gelöscht.
```

Laden, Speichern und Löschen

Zum Abschluss des Kapitels fehlen noch Befehle, um die Variablen im Workspace zu verwalten und späteren Arbeiten zur Verfügung zu stellen. Im einzelnen sind dies:

- **clear**: Löscht alle Variablen im Workspace
- **clear *variable***: Löscht *variable* aus dem Workspace
- **save**: Speichert alle Variablen in Datei 'matlab.mat'
- **save *fname***: Speichert alle Variablen in Datei 'fname.mat'
- **save *fname* x y z**: Speichert Variablen x, y und z in Datei 'fname.mat'
  und entsprechend
- **load**: Lädt alle Variablen aus Datei 'matlab.mat' in den Workspace
- **load *fname***: Lädt alle Variablen aus Datei 'fname.mat' in den Workspace
- **load *fname* x y z**: Lädt Variablen x, y und z in den Workspace

Von diesen Befehlen sind hier die wichtigsten Optionen vorgestellt, sie besitzen aber noch weitere. Im allgemeinen sei mit dieser Einführung immer wieder auf die MATLAB-Hilfe verwiesen, um alle Möglichkeiten kennenzulernen und auszunutzen.

Aufgaben und Übungen

1.6 ☞ Erstellen Sie zwei Variablen $\mathbf{t}= \begin{bmatrix}2 & 4 & 6 & \dots & 14\end{bmatrix}$, $\mathbf{y}= \begin{bmatrix}1 & 5 & 7\\ 2 & 5 & \pi\end{bmatrix}$ und verändern Sie ihren Inhalt durch einen Doppelklick auf die Variable im Workspace-Fenster. Überprüfen Sie die Änderung, indem Sie die Variablennamen im Befehlsfenster eingeben.

1.7 ☞ Ändern Sie den Namen von **y** auf **ymat**.

1.8 ☞ Erstellen Sie einen Zeilenvektor **t1** mit Werten von 0 bis 1 und dem Abstand 0.01.

1.9 ☞ Erstellen Sie einen Spaltenvektor **t2** mit logarithmischer Teilung und Werten von 10^{-3} bis 10^{0}.

1.10 ☞ Extrahieren Sie die erste Zeile **Z1** und die dritte Spalte **C3** von **ymat**.

1.11 ☞ Extrahieren Sie die Matrix **yklein** mit der ersten und dritten Spalte von **ymat**.

1.12 ☞ Extrahieren Sie den Vektor **zwischen** mit Werten größer als 3 aus der Matrix **ymat**.

1.13 ☞ Löschen Sie die zweite Zeile von **ymat**.

1.14 ☞ Speichern Sie alle Variablen in eine Datei und löschen Sie den Workspace mit **clear all**. Überprüfen Sie den Inhalt des Workspace (z. B. **who**). Laden Sie Ihre Variablen aus der Datei erneut in den Workspace.

1.3 Operatoren und Funktionen

Operatoren

Die rechte Seite einer Zuweisung darf ein beliebig komplizierter algebraischer Ausdruck sein. Konstanten und bekannte Variablen dürfen in sinnvoller Weise mit Operatoren verknüpft werden. Bei den arithmetischen Operatoren sind neben den üblichen (+,-,*,/) insbesondere das Potenzieren (^) und Transponieren (') von Bedeutung. Daneben gibt es auch die logischen Operatoren ~ (nicht), & (und), | (oder), xor() (exklusiv oder) und die relativen Operatoren == (gleich), ~= (ungleich), <, ≤, >, ≥. Es gelten die üblichen Regeln für Punkt- vor Strichrechnungen. Andere Gruppierungen können mit runden Klammern ( ) erreicht werden. Eine Übersicht über die Operatoren liefert der Befehl:

```
» help ops
```

Achtung: Die Operatoren beziehen sich auf Matrixberechnungen. Möchte man den Operator auf die einzelnen Elemente einer Matrix anwenden, so muss dem Operator ein Punkt vorangestellt werden!

```
» A = [1, 2, 3; 4, 5, 6; 7, 8, 0]

» b                    % Zeilenvektor
» b=b'                 % Spaltenvektor
» b+b                  % Vektorsumme
» b'*b                 % Skalarprodukt
» b*b                  % Fehler, weil inkompatible Dimension
» b.*b                 % elementweise Multiplikation
» 2*A                  % Multiplikation mit einem Skalar
» A*A                  % A mal A und
» A ^ 2                % A hoch 2 sind identisch
» A .^ 2               % aber nicht hiermit!!!
```

Funktionen

Wie schon implizit in den vorigen Abschnitten gesehen, enthält MATLAB sehr viele Funktionen aus den verschiedensten Gebieten. Ein Funktionsaufruf besteht aus einem Namen und Übergabeparametern. Das Ergebnis kann einer Variablen zugewiesen werden. Eine Übersicht über mathematische Standardfunktionen liefert der Befehl **help elfun**.

```
» help elfun
» cos (0)
» 4 * atan(1)        % pi, atan(x) = tan^-1(x)
» bexp = exp(b)      % Exponentialfunktion jedes einzelnen Elements von b
```

Zwei sehr nützliche Funktionen sind **size** und **length**. **size** gibt die Anzahl an Zeilen und Spalten einer Matrix an und ermöglicht somit, Zeilen- von Spaltenvektoren zu unterscheiden. **length** gibt die Länge eines Vektors an und im Falle einer Matrix den größeren Wert der Spalten- und Zeilenanzahl.

```
» A=[1, 2, 3; 4, 5, 6]
» size(A)
» length(A)
» size(b)
» length(b)
```

Eine Übersicht über Standardmatrizen und Matrizenmanipulationen liefert der Befehl **help elmat** sowie über Funktionen aus dem Bereich der linearen Algebra **help matfun**. Die Funktionen **ones** und **zeros** zum Beispiel erstellen Matrizen mit nur 1 bzw. 0 als Einträgen und die Funktion **eye** erstellt die Einheitsmatrix.

```
» help elmat
» D =ones(3,4)
» E =zeros(5)
» F =eye(4)
» G =eye(4,3)
» help matfun
```

Aufgaben und Übungen

1.15 ☞ Geben Sie die Anzahl der Zeilen und Spalten von D, E, F und G an.

1.4 Graphische Darstellungen

MATLAB besitzt ebenfalls eine Reihe von graphischen Möglichkeiten. Die wichtigsten Graphikfunktionen sind:

- 2D-Kurven: **plot, fplot, ezplot, subplot, stem, stairs**
- 3D-Kurven: **plot3, stem3, surf, mesh, contour**

Der Befehl 'plot'

Möchte man einen Vektor **y** graphisch darstellen, so lautet der Befehl **plot(y)**. In dem Fall werden die Elemente von **y** (y_1, y_2,y_3,...) über ihre Indizes (1,2,3...) 2-dimensional aufgetragen. Möchte man eine andere x-Achse, z. B die Zeit t, so muss dieser Vektor **t** dem Vektor **y** vorangestellt werden, **plot(t,y)**. Der entsprechende Befehl für die dreidimensionale Darstellung, bei dem immer alle drei Vektoren angegeben werden müssen, lautet **plot3(t,x,y)**.

```
» t = linspace(-pi, pi, 30);
» y1= sin(2 *t) ;
» y2= cos(5 *t) ;
» plot(y1)
» plot(t, y1)                % Zum Vergleich
» plot(y1, t)                % Zum Vergleich
» plot3(t, y1, y2)
```

Die Zeichnungen werden in sogenannten **figure**s dargestellt. Dies sind eigene Fenster, die neben der Kurve noch die Achsen, verschiedene Menübefehle zum Ändern des Layouts, ... enthalten. Ist bereits ein figure-Fenster offen, so wird die Kurve in dieses Fenster geplottet und die alte Kurve gelöscht. Mit **hold** wird/werden die alte(n) Kurve(n) beibehalten und die neue dazu geplottet. Ist noch kein figure-Fenster offen oder der Befehl **figure** dem plot-Befehl vorangestellt, wird ein neues Fenster erzeugt.

Die meisten Layout-Einstellungen sind nicht nur über das figure-Menü, sondern auch über das Befehlsfenster zu ändern. Nützliche Befehle sind:

```
» grid on/off    % schaltet ein Hintergrundraster (grid) dazu oder weg
» box on/off     % schaltet eine umrahmende Box dazu oder weg
» axis on/off    % schaltet alle aktuellen Achsendarstellungen hinzu oder weg
```

Wird on/off bei den ersten beiden Befehlen weggelassen, so wird zwischen den beiden Zuständen (an/aus) hin- und hergeschaltet.

Der Plotbefehl lässt ein zusätzliches Argument, eine Zeichenkette, zu, z. B. **plot(t,y1,'g')**. In dieser Zeichenkette können Angaben zur Farbe, Liniendarstellung und Punktedarstellung gemacht werden. Für eine genaue Beschreibung sowie weiterer Möglichkeiten rufen Sie bitte die Hilfe mit **help plot**, **help graph2d** und **help graph3d** auf.

Ein figure-Fenster kann wie eine Matrix in mehrere „Unter-Bilder“mit dem Befehl **subplot** aufgeteilt werden:

```
» figure                          % öffnet ein neues figure-Fenster
» subplot(2,1,1); plot(t,y1)      % subplot teilt das Fenster in 2 Zeilen und 1 Spalte auf
                                  % und wählt das erste Element (es wird von links nach rechts
                                  % und von oben nach unten gezählt) zur Darstellung der
                                  % nächsten geplotteten Kurve aus
» subplot(2,1,2); plot(y1,t)      % subplot benutzt die gleiche Aufteilung, wählt aber jetzt
                                  % das zweite Fenster aus
```

Zur einfachen Beschriftung existieren folgende Funktionen, welche auf den gerade aktuellen Graphen angewendet werden:

```
» title('Schöne Kurve')    % überschreibt den Graph mit einem Titel
» xlabel('Zeit /s')        % beschriftet die x-Achse
» ylabel('z /m')           % beschriftet die y-Achse
```

Enthält ein Graph mehrere Kurven, kann mit dem Befehl **legend('Kurve1','Kurve2',...)** eine Legende erzeugt werden.

Andere Graphen

Die Funktionen **stem** bzw. **stem3** und **stairs** sind für die Darstellung diskreter Signale geeignet. Dabei stellt stem jeden Punkt durch einen senkrechten Balken mit einem Kreis an der Spitze dar. Stairs ist die Treppenfunktion und verbindet die einzelnen Punkte wie eine Treppe, indem der Wert des vorigen Punkts bis zum nächsten beibehalten wird.

fplot berechnet in einem angegeben Intervall die Werte und den Kurvenverlauf einer Funktion, ohne vorher die genaue Anzahl an Punkten anzugeben.

```
» stem(y1)
» stairs(t,y1)
» fplot('3* sin(r).* exp(-r/(pi/4))', [0, 2* pi]);
```

Darstellung von Oberflächen

Zur Darstellung von Oberflächen werden die Befehle **mesh** und **surface** benutzt:

- **mesh** plottet ein buntes Netz, bei dem die Knoten die Punkte der angegebenen Funktion sind.
- **surf** stellt auch die Maschen des Netzes, also die gesamte Oberfläche, bunt dar.

Die Farbenpalette wird über **colormap** festgelegt und kann mit Hilfe **colorbar** angesehen werden.

Anhand eines Beispiels wird verdeutlicht, wie die benötigten Vektoren erzeugt werden. Wir möchten die Oberfläche des Graphen der Funktion $z = -2x + 3y$ darstellen, wobei x und y die folgenden Werte annehmen sollen:

x = [-1, 0, 1, 2]
y = [0, 0.1, 0.2, 0.3, 0.4, 0.5]

Da **x** 4 und **y** 6 Werte enthält, muss **z** $6 \times 4 = 24$ Werte annehmen. Darüber hinaus müssen, da jeder Punkt durch ein Tripel (x_i, y_i, z_i) beschrieben wird, die 6×4-Matrizen **X** und **Y** erzeugt werden, die in jeder Zeile bzw. Spalte die Werte von **x** und **y** enthalten. Dies geschieht am einfachsten mit dem Befehl **meshgrid**. Anschließend kann **z** berechnet und alles gezeichnet werden:

```
» x = [-1 : 2]; y = [0 : 0.1 : 0.5]    % Definition der Vektoren x und y
» [X, Y] = meshgrid (x,y);             % Erzeugen der Matrizen X und Y
» Z = -2.* X + 3.* Y;                  % Berechnen der Matrix Z
» mesh ( X, Y, Z)                      % oder auch surf ( X, Y, Z)
```

Abschließend siehe auch die Befehle: **compass, contour, contour3, surfc, waterfall, pcolor, view**

Aufgaben und Übungen

1.16 ☞

Erzeugen Sie die Matrix **graphA**= $\begin{bmatrix} 1 & 2 & 3 & \dots & 6 \\ 1 & 4 & 9 & \dots & 36 \\ 1 & 8 & 27 & \dots & 216 \end{bmatrix}^{\mathbf{T}}$ und plotten Sie **graphA**. Fügen Sie dem Graph eine Legende mit den Texten „linear“, „quadratisch“ und „kubisch“ hinzu.

Schalten Sie das Hintergrundraster ein. Stellen sie über das Menü **Edit → Axes Properties** die Achsenskalierung der y-Achse auf logarithmisch um. Geben Sie der Kurve den Titel „graphA“.

1.17 ☞ Plotten Sie die Funktion $c = y * sin(x)$ für x zwischen -10 und 10 (Inkrement 1) und y zwischen 0 und 30 (Inkrement 3) und stellen Sie danach ihre Oberfläche als Netz dar.

1.18 ☞ Berechnen Sie per Hand die Übergangsfunktion eines Verzögerungssystems 1. Ordnung mit dem Übertragungsfaktor 3 und der Verzögerungszeit 2. Stellen Sie die Übergangsfunktion graphisch dar.

1.5 Probleme der numerischen Darstellung

Ein lineares Gleichungssystem $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$ kann man anhand der Inversen der Matrix **A** lösen: Falls **A** regulär ist, gilt nämlich

$$A \cdot x = b \Rightarrow x = A^{-1} \cdot b$$

Wie man weiß, können reelle Zahlen nur mit endlicher Stellenzahl auf einem Rechner dargestellt werden. Sie werden also abgerundet. Dies geschieht auch mit dem Ergebnis jeder Operation von Zahlen, wenn das genaue Ergebnis nicht mehr innerhalb der Grenzen der Gleitpunktdarstellung passt.
Welche Wirkung dieser Fehler auf die Berechnung einer Matrizen-Inverse oder bei der Bestimmung der Lösung eines Gleichungssystems hat, soll hier anhand eines Beispiels angedeutet werden.

Geben Sie die Matrix

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

ein und berechnen Sie deren Inverse mittels des Befehls **inv**. Berechnen Sie dann die Determinante dieser Matrix, indem Sie den Befehl **det** benutzen.

Was schließen Sie aus den Ergebnissen ?

Obwohl die Determinante der Matrix A Null ist (exakte Lösung) gibt uns MATLAB trotzdem eine Inverse für **A**. Um dieses Ergebnis weiter zu überprüfen, berechnen Sie das Produkt aus der Matrix und ihrer Inversen:

≫ A ∗ inv(A)

Das Ergebnis ist offensichtlich nicht gleich der Einheitsmatrix !!

Was ist die Erklärung dafür ? Der Algorithmus, welcher den Befehl **inv** aufruft, wird auf die Elemente der Matrix **A** eine Reihe von elementaren Operationen anwenden und gewisse Ergebnisse auswerten müssen, um die Aussage über die Invertierbarkeit der Matrix treffen zu können.

Gehen Sie zunächst davon aus, dass die folgenden Operationen durchgeführt werden würden

$$\begin{aligned} x &= 0.1 \\ y &= 1000 \\ a &= 0.1 \\ b &= y + a \\ c &= b - y \\ d &= c - a \end{aligned}$$

und je nachdem, ob d verschwindet oder nicht, entscheidet der Algorithmus, ob die Matrix **A** singulär ist oder nicht. Wie man leicht nachrechnen kann, ist hier $d = 0$. Um nachzuprüfen, welches Ergebnis MATLAB liefert, geben sie die oben angegebenen Operationen im Befehlsfenster ein.

Anscheinend ist d ungleich Null. Also würde MATLAB an dieser Stelle den Algorithmus nicht abbrechen, sondern weiter die Inverse berechnen, was zu einem falschen Ergebnis führen würde.

MATLAB hatte uns aber vorher gewarnt. Bei der Berechnung der Inversen von **A** erschien nämlich die Warnung

```
Warning: Matrix is close to singular or badly scaled.
 Results may be inaccurate. RCOND = 2.055969e-018.
```

Die zweite Aussage trifft mit Sicherheit in diesem Fall zu !
Was die erste bedeutet und wofür **RCOND** steht, soll anhand eines weiteren Beispiels [1] gezeigt werden.

Bestimmen Sie die Lösung folgenden Gleichungssystems:

$$A \cdot \mathbf{x} = b$$

mit

$$A = \begin{bmatrix} 10 & 7 & 8 & 7 \\ 7 & 5 & 6 & 5 \\ 8 & 6 & 10 & 9 \\ 7 & 5 & 9 & 10 \end{bmatrix} \quad b = \begin{bmatrix} 32 \\ 23 \\ 33 \\ 31 \end{bmatrix}$$

Die exakte Lösung lautet x=$\begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}^T$.

Betrachtet wird nun die Empfindlichkeit der Lösung **x** gegenüber Störungen in den Koeffizienten von A und b. Dazu wird der relative Fehler einer Matrix **y** (mit der veränderten Matrix $\mathbf{y}_1$) eingeführt:

$$er_y = \max\left(\left|\frac{y_{ij} - y_{1ij}}{y_{ij}}\right|\right)$$

Man wähle zunächst $\mathbf{b}_1 = \begin{bmatrix} 32.1 & 22.9 & 33.1 & 30.9 \end{bmatrix}^T$. Berechen Sie den relativen Fehler von **b**. Bestimmen Sie die Lösung $\mathbf{x_1}$ von $A{\cdot}x_1 = b_1$ (die exakte Lösung lautet: $\mathbf{x}_1 = \begin{bmatrix} 9.2 & -12.6 & 4.5 & -1.1 \end{bmatrix}^T$) und den relativen Fehler von **x**. Wie lautet Ihr Fazit?

Als nächstes werden die Koeffizienten der Matrix **A** geringfügig verändert:

$$A_1 = \begin{bmatrix} 10 & 7 & 8.1 & 7.2 \\ 7.08 & 5.04 & 6 & 5 \\ 8 & 5.98 & 9.89 & 9 \\ 6.99 & 4.99 & 9 & 9.98 \end{bmatrix}$$

Berechnen Sie den relativen Fehler von **A** und die Lösung $\mathbf{x_2}$ des Gleichungssystems $A_1 \cdot x = b$ (exakte Lösung: $\mathbf{x}_2 = \begin{bmatrix} -81 & 137 & -34 & 22 \end{bmatrix}^T$.
Berechnen Sie den relativen Fehler von **x**.

[1] aus Larrouturou, Lions: Optimisation et commande optimale. Ecole Polytechnique

Fazit: Kleine Änderungen in den Koeffizienten der Matrix **A** oder in **b** verursachen erhebliche Änderungen in der Lösung des Gleichungssystems $A \cdot x = b$. Man sagt, die Matrix **A** ist von schlechter Kondition und ein Maß dafür ist die Konditionszahl $\kappa(A)$, welche folgende Eigenschaft besitzt:

$$1 \leq \kappa(A)$$

Je näher diese Zahl bei 1 ist, desto besser ist die Matrix A konditioniert und man kann erwarten, dass MATLAB richtige Ergebnisse bei der Inversion oder bei der Lösung von Gleichungssystemen liefert.
Genau das Gegenteil ist der Fall, wenn die Konditionszahl sehr groß ist.

Ist diese sogar so groß, dass die Rechnergenauigkeit nicht mehr genügt, um sinnvolle Ergebnisse zu liefern, warnt MATLAB

```
Warning: Matrix is close to singular or badly scaled.
 Results may be inaccurate. RCOND = 2.055969e-018.
```

wobei RCOND die Inverse der Konditionszahl von **A** ist, d.h. je näher RCOND an Null ist, desto schlechter ist die Matrix konditioniert.

Wären **A** und **b** das Ergebnis von vorher auf dem Rechner durchgeführten Kalkulationen, so hätten wir wahrscheinlich nicht die richtige Lösung getroffen !!!! Hier liefert MATLAB die richtige Lösung, weil die Fehlerabweichung im Rahmen der Rechnergenauigkeit von MATLAB bleibt.

Was kann man unternehmen, um das Ergebnis zu verbessern?

1. Wenn man mit single precision arbeitet, sollte man auf jeden Fall die Genauigkeit in double precision (Standard bei MATLAB) umstellen.

2. Meistens benötigt man nicht explizit die Inverse einer Matrix. Zur Lösung des Gleichungssystems

   $$A \cdot x = b$$

   benutze man dann anstatt x = inv(A) ∗ b den Befehl der Linksdivision A\ b . Dieser beruht auf dem Gauss Algorithmus, welcher keine Invertierung einer Matrix benötigt und generell stabil ist. Im Fall überbestimmter Gleichungssysteme liefert er die im Sinn der kleinsten Fehlerquadrate (least squares) beste Lösung.

3. Bei Matrizen, welche besondere Eigenschaften besitzen, z.B. symmetrisch oder positiv definit sind (u. a. der Fall bei Gleichungssystemen in der Parametrischen Identifikation von linearen zeitinvarianten Systeme), werden Algorithmen benutzt, welche diese Eigenschaft in Betracht ziehen (z. B. Cholesky Verfahren).

Fazit: Ziel dieses Abschnitts war nicht, Angst in dem Umgang mit MATLAB einzujagen. MATLAB liefert bei den meisten Berechnungen richtige Ergebnisse!
Das Ziel war vielmehr, auf gewisse Aspekte der numerischen Mathematik hinzuweisen, das Bewusstsein dafür zu schärfen und auch die Möglichkeit zu geben, mit MATLAB-Warnungen in dieser Richtung umzugehen.

Aufgaben und Übungen

1.19 ☞ Berechnen Sie die Lösung des Gleichungssystems

$$\begin{array}{llll} 1 \cdot x_1 & +2 \cdot x_2 & +3 \cdot x_3 & = 402 \\ 4 \cdot x_1 & +2 \cdot x_2 & +1 \cdot x_3 & = 521 \\ 7 \cdot x_1 & +5 \cdot x_2 & +9 \cdot x_3 & = 638 \end{array}$$

2 Programmieren mit Matlab

M.File

Im ersten Abschnitt wurde bereits erläutert, wie eine neue Datei mit dem MATLAB Editor/Debugger erzeugt oder eine bestehende verändert werden kann. Programme in MATLAB müssen in Dateien mit der Endung '.m' abgespeichert werden, damit sie als neue Befehle erkannt werden. Darüber hinaus müssen die Dateien entweder im aktuellen Arbeitsverzeichnis oder in einem Verzeichnis des Suchpfads enthalten sein. Unter dieser Voraussetzung werden die Dateien durch Eingabe ihres Namens (bei Funktionen unter Angabe der benötigten Argumente), wie von MATLAB mitgelierte Befehle auch, ausgeführt.

Skripte und Funktionen

Skripte sind Dateien, die eine Folge von MATLAB-Befehlen enthalten, genauso, als wären sie direkt im Befehlsfenster geschrieben worden. Die verwendeten Variablen sind ebenso im Workspace enthalten. In Aufgabe 1.5 am Ende des ersten Abschnitts wurde sogar bereits ein kleines Skript erstellt.

Funktionen besitzen einen eigenen Workspace mit lokalen Variablen und haben Ein- und Ausgangsargumente. Neben selbstgeschriebenen können auch bereits mit MATLAB/SIMULINK mitgelieferte Funktionen, die in MATLAB (und nicht z. B. C) geschrieben sind, editiert werden, z. B. **edit factorial**. Folgende Syntax muss bei einer Funktion beachtet werden:

```
function [Ausgangsargumente] = funktionsname(Eingangsargumente)
% Alles, was hier angegeben wird, erscheint, falls man die Hilfe zu
% dieser Funktion mit 'help funktionsname' aufruft.

Liste der funktionseigenen Befehle
```

Der angegebene 'funktionsname' muss dabei mit dem Namen der Datei 'funktionsname.m' übereinstimmen. Die erste Zeile darf nicht mit einem Semikolon abgeschlossen werden. Die zweite und folgenden anschließenden Zeilen, die mit einem % beginnen, werden bei Aufruf der Hilfe zu dieser Funktion ausgegeben.

Zur Verdeutlichung folgt ein einfaches Beispiel. Schreiben Sie folgende Anweisungen in eine Datei mit dem Namen 'lingls.m':

```
function [x, detA] = lingls(A,b)
% LINGLS loest das lineare Gleichungssystem A*x=b und gibt
% den Loesungsvektor x sowie die Determinante der Matrix A zurueck.

detA=det(A); x=A\b;
```

Führen Sie anschließend die Befehle aus:

```
» help lingls
» matA = [1, 2, 3; 4, 5, 6; 7, 8, 0];
» vek = [2; 5; 9]
» lingls(matA, vek)
» [antw1, antw2] = lingls(matA, vek)
```

Möchte man eine Funktion schreiben, die mit einer unterschiedlichen Anzahl an Argumenten aufgerufen werden kann, sind die MATLAB-Funktionen **nargin** und **nargout** hilfreich. Sie liefern die Anzahl der Ein- und Ausgangsargumente. Ihre Verwendung ist in der Funktion bode gut zu erkennen: **edit subplot**.

Möchte man die in einer Funktion verwendeten Variablen, z. B. t und x, auch anderen Funktionen zur Verfügung stellen oder im Workspace sichtbar machen, müssen sie an jeder Stelle (d. h. in den entsprechenden anderen Funktionen, aber auch im Workspace) als **global** deklariert werden:

```
» global t, x
```

Dies stellt aber programmiertechnisch eine unschöne Lösung dar. Die bessere Programmierung ist die Aufnahme als Ausgangsargument.

Wie in jeder anderen Programmiersprache auch, besitzt MATLAB Kontrollstrukturen. Diese sind

- **if** ... **elseif** ... **else** ... **end**
- **switch** ... **case** ... **otherwise** ... **end**
- **for** ... **end**
- **while** ... **end**
- **try** ... **catch** ... **end**

Benutzen Sie für die Syntax die entsprechende Hilfefunktion.

Aufgaben und Übungen

2.1 ☞ Schreiben Sie eine Funktion, die

- als Eingangsargumente zwei Vektoren erhält,
- überprüft, ob es sich wirklich um Vektoren und nicht um Matrizen handelt
- im Falle der Eingabe einer Matrix eine Fehlermeldung in dem Befehlsfenster ausgibt und den Wert Null als Ausgangs-Argument zurückgibt
- sonst den größten Wert beider Vektoren berechnet und diesen als Ausgangs-Argument zurückgibt
- Verfassen Sie dazu einen entsprechenden Hilfe-Text.

2.2 ☞ Schreiben Sie eine Funktion, die den Wert der abschnittsweise definierten Funktion in Abhängigkeit von t berechnet:

$$f(t) = \begin{cases} 0 & \text{für } t < 0 \\ t^2 \cdot (3 - 2 \cdot t) & \text{für } 0 \leq t \leq 1 \\ 1 & \text{für } t > 1 \end{cases}$$

Stellen Sie die Funktion im Bereich von -1 bis 3 graphisch dar.

2.3 ☞ Gegeben sei ein lineares, zeitinvariantes System durch seine Differentialgleichung

$$my'' + fy' + ky = u$$

und der Anfangsbedingungen $y(0) = y'(0) = 0$. Geben Sie in Abhängigkeit von m, f und k die Kennkreisfrequenz, den Dämpfungsgrad, die Eigenkreisfrequenz und den statischen Übertragungsfaktor an. Schreiben Sie eine Funktion, die als Eingangsargumente m, f und k erhält und diese Berechnungen durchführt und die Werte zurückgibt. Führen Sie diese Funktion für die Werte k=3, m= 5 und f=4 aus.

3 Matlab in der Regelungstechnik

MATLAB selbst ist eine Rechenumgebung, die die Entwicklung eigener Algorithmen ermöglicht. Bereits entwickelte Algorithmen werden nach Themengebieten geordnet in sogenannten **Toolboxen** angeboten. So existieren Toolboxen z. B. zur Signalverarbeitung, Identifikation, Reglerentwurf aber auch zur Statistik, Optimierung, ... Die Liste ist sehr umfangreich und wächst ständig.

Im Bereich der Regelungstechnik ist die Basis die **Control System Toolbox**, die hier auch vorgestellt werden soll. Weitere am Institut für Regelungstechnik vorhandene Toolboxen von Bedeutung sind (ohne Anspruch auf Vollständigkeit) die **Model Predictive Control Toolbox, Signal Processing Toolbox, System Identification Toolbox, Optimization Toolbox, LMI Control Toolbox, Robust Control Toolbox, Wavelet Toolbox** und **Neural Network Toolbox**.

Control System Toolbox

Die Control System Toolbox kennt neue Datenstrukturen zur Darstellung linearer, zeitinvarianter Systeme (Linear Time Invariant: LTI-Systems), die Modelle. Sie werden über die drei parametrischen Funktionen **tf** (Übertragungsfunktion: Transfer Function), **zpk** (Pol-/Nullstellen-/Übertragungsfaktor: Zero/Pole/Gain)), **ss** (Zustandsraum: State Space) und die nichtparametrische Funktion **frd** (Frequenzgang: Frequency Response Data) erzeugt. Im folgenden werden allein zur besseren Übersicht die entsprechend definierten Variablen **sys**$*$ genannt.

Die Übertragungsfunktion in MATLAB benötigt als Argumente für **tf** die Angabe zweier Zeilenvektoren, die die Koeffizienten von Zähler- und Nennerpolynom in fallender Potenz von s enthalten. Für eine Übertragungsfunktion

$$G(s) = \frac{3s - 2}{0.25s^2 + 0.1s + 1}$$

folgt somit

```
» z = [3 -2];
» n = [0.25 0.1 1];
» sys1 = tf(z,n)
```

Bei der Darstellung von dynamischen Systemen durch die Zustandsraumdarstellung wird das Systemverhalten über gekoppelte Differentialgleichungen 1. Ordnung beschrieben,

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \\ \mathbf{y} &= \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} \end{aligned}$$

und in Form der vier Matrizen **A**,**B**,**C** und **D** in MATLAB abgespeichert. Mit **x** wird der Vektor der Zustandsgrößen bezeichnet. Für MIMO (**M**ulti-**I**nput-**M**ulti-**O**utput)-Systeme können die Ein- und Ausgangsgrößen **u** und **y** auch in vektorieller Form auftreten.

Die Transformation der oben vorgestellten Übertragungsfunktion in die Zustandsraumdarstellung, d. h. die Berechnung und Darstellung der vier Matrizen **A**,**B**,**C** und **D** aus den Zeilenvektoren des Zähler- und Nennerpolynoms des Übertragungsfunktions-Modells **sys1**, ist sehr einfach: Der Befehl **ss** muss nur als Argument das vorher definierte Modell **sys1** erhalten:

```
» sys2 = ss(sys1);   %
```

Der Vollständigkeit halber sei noch kurz die Syntax von **zpk** und **frd** aufgeführt. Für ausführliche Erläuterungen rufen Sie bitte die Hilfe auf.

```
» z = 0.5; p = [-1, 2]; k = 3;          % Nullstelle bei 0.5, Polstellen bei -1 und 2,
                                        % Übertragungsfaktor 3
» sys3 = zpk(z, p, k)
» fgang= [3-i, 8-4i, 2-3i]; freq = [0.1, 1, 10];   % Werte des Frequenzgangs
                                        % an den Frequenzen 0.1, 1 und 10 rad/s
» sys4 = frd ( fgang, freq);            %
```

Allgemein erfolgt eine Umwandlung zwischen den verschiedenen Darstellungsformen eines Modells automatisch, indem den Befehlen **tf, ss** oder **zpk** allein ein in anderer Darstellung definiertes parametrisches Modell (nicht frd!) als Argument gegeben wird. Einzige Ausnahme bildet die nichtparametrische Darstellung in **frd**. Sie wird aus den anderen mit einer zusätzlichen Angabe über den Frequenzvektor erzeugt. Umgekehrt können aber die parametrischen Darstellungen nicht mehr aus frd gewonnen werden!

Die den vier Funktionen zur Erzeugung von Modellen angegebenen Argumente können aus den Modellen **sys**$*$ wieder mit Hilfe der Funktionen **tfdata, ssdata, zpkdata** und **frdata** gewonnen werden. Näheres hierzu in der MATLAB-Hilfe.

Bisher wurde davon ausgegangen, dass es sich um die Darstellung zeitkontinuierlicher Systeme handelt. Analog zur Differentialgleichung im zeitkontinuierlichen existiert für den zeitdiskreten Bereich eine das dynamische Systemverhalten beschreibende Differenzengleichung:

$$\alpha_m y_{k-m} + \cdots + \alpha_1 y_{k-1} + \alpha_0 y_k = \beta_m u_{k-m} + \cdots + \beta_1 u_{k-1} + \beta_0 u_k$$

Die diskrete Übertragungsfunktionen lautet

$$G(z) = \frac{\beta_m z^{-m} + \cdots + \beta_1 z^{-1} + \beta_0}{\alpha_m z^{-m} + \cdots + \alpha_1 z^{-1} + \alpha_0} = \frac{\beta_m + \cdots + \beta_1 z^{m-1} + \beta_0 z^m}{\alpha_m + \cdots + \alpha_1 z^{m-1} + \alpha_0 z^m}$$

und die diskrete Zustandsraumbeschreibung:

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{A}_d \mathbf{x}_k + \mathbf{B}_d \mathbf{u}_k \\ \mathbf{y}_k &= \mathbf{C}\mathbf{x}_k + \mathbf{D}\mathbf{u}_k \end{aligned}$$

Anknüpfend an die kontinuierlichen Beispiele lassen sich zeitdiskrete Systeme durch das zusätzliche Argument einer Abtastzeit direkt angeben

```
» Tabt = 1                              % Abtastzeit Tabt = 1s
» sys1dd = tf(z,n,Tabt)
» sys2dd = ss(A, B, C, D, Tabt);        % zpk und frd funktionieren analog
```

oder mit Hilfe des Befehls **c2d** (continuous to discrete) aus den kontinuierlichen Modellen gewinnen. Der Befehl **d2d** (discrete to discrete) ermöglicht eine spätere Änderung der Abtastzeit und der Befehl **d2c** (discrete to continuous) erzeugt wieder ein kontinuierliches Modell.

```
» sys1d = c2d(sys1, Tabt)
» sys2d = c2d(sys2, Tabt);
» Tabt2 = 2                             % Abtastzeit Tabt2 = 2s
» sys1d2 = d2d(sys1d, Tabt2)
» sys1c = d2c(sys1d)                    % Vergleiche mit sys1
```

Warum unterscheiden sich die beiden zeitdiskreten Modelle 'sys1dd' und 'sys1d' sowie 'sys2dd' und 'sys2d'?

Die bisher in diesem Kapitel gezeigten Darstellungsformen dynamischer Systeme sind zum besseren Verständnis teilweise in der folgenden Graphik dargestellt. Die Funktionen und Darstellungsformen aus MATLAB sind grau hinterlegt.

Dynamische Systeme	**kontinuierlich**		**diskret**	
Übertragungsfunktion (transfer function)	$G(s) :=$ $tf(\mathbf{z}, \mathbf{n})$	**c2d**	$tf(\mathbf{z_d}, \mathbf{n_d}, \mathbf{T_{abt}})$	$:= G(z)$
	ss $\Downarrow \Uparrow$ **ff**	$\rightleftharpoons$	**ss** $\Downarrow \Uparrow$ **tf**	
Zustandsraum (state space)	$ss(\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D})$	**d2c**	$ss(\mathbf{Ad}, \mathbf{Bd}, \mathbf{C}, \mathbf{D}, \mathbf{T_{abt}})$	
	$\Updownarrow$ gekop.Dgln 1. O.		$\Updownarrow$ gekop.Δgln 1. O.	

mit den Abkürzungen: *Dgl:* *Differentialgleichung*
Δ*gl:* *Differenzengleichung*
gekop.: *gekoppelt*

Totzeitsysteme können bei einem Modell *sys* mit Hilfe der Eigenschaft **'iodelaymatrix'** und anschließender Angabe der Dauer der Totzeit eingeführt oder verändert werden. Wenn das Modell *sys* bereits existiert, kann diese Eigenschaft mit dem Befehl **set** verändert werden. Existiert es noch nicht, wird die Eigenschaft bei der Erzeugung mit angegeben. Im zeitdiskreten Fall gibt die Totzeit die Anzahl der Abtastschritte an, sie muss also ganzzahlig sein.

```
» set(sys1,'iodelaymatrix', 0.5); sys1      % Totzeit von 0.5s in sys1
» set(sys2d,'iodelaymatrix', 4); sys2d      % Totzeit von 4 Abtastschritten
                                            % in sys2d
» systot = zpk(0.5, [-1, -2], 2.5, 'iodelaymatrix', 0.75);
» systotd = zpk(0.5, [-1, -2], 2.5, 4, 'iodelaymatrix', 2);
```

Nachdem die Erzeugung und Umwandlung dynamischer Systeme in MATLAB behandelt worden sind, geht es nun um die graphische Darstellung und Charakterisierung der Systeme.

Für die Berechnung und Darstellung der Antwort des Systems im Zeitbereich stehen die Funktionen **impulse** (Gewichtsfunktion=Antwort auf einen Einheits-Impuls), **step** (Übergangsfunktion=Anwort auf einen Einheits-Sprung) und **lsim** (Antwort auf ein beliebiges Eingangssignal) zur Verfügung. Werden diese Befehle ohne linksseitige Argumente aufgerufen, so wird die berechnete Antwort nicht im Befehlsfenster ausgegeben sondern geplottet.

```
» [h, th]= step(sys1);
» [g, tg] = impulse(sys1);
» plot(th, h, tg, g);
» step(sys2d);
» t=[0: 0.1: 10];
» u= ones(size(t));
» u(41:70) = 0*ones(30,1);
» lsim(sys1, u, t)
```

Für die Darstellung des Frequenzgangs eines Systems gibt es ebenfalls mehrere Möglichkeiten. Die

aus der Vorlesung MRT bekannten sind das Bode-Diagramm (**bode**) und die Ortskurvendarstellung (**nyquist**), auch Nyquist-Diagramm genannt.

```
» bode(sys1);
» nyquist(sys1);
```

Weitere Möglichkeiten sind die Befehle **nichols**, **sigma** und **freqresp** (siehe die MATLAB-Hilfe für weitere Erläuterungen).

Eine sehr angenehme Möglichkeit zur übersichtlichen Darstellung und Analyse dynamischer Systeme bietet die graphische Oberfläche LTI VIEWER, die mit dem Befehl **ltiview** aufgerufen wird. Sie ermöglicht die gleichzeitige Darstellung mehrerer Systeme sowie mehrerer Darstellungen nebeneinander und der einfachen Umschaltung zwischen verschiedenen Darstellungen. Diese Oberfläche ist größtenteils selbsterklärend, also probieren Sie sie einfach aus!

Eine Übersicht aller Befehle der Control System Toolbox liefert der Befehl **help control**. Sehen Sie aber auch die Befehle **ltimodels** und **ltiprops** sowie aus der Signal Processing Toolbox **tf2ss** und **zp2ss**.

Aufgaben und Übungen

3.1 ☞ Erstellen Sie zwei Modelle, **modell1** und **modell2**. Das erste soll eine Übertragungsfunktion enthalten, das zweite im Zustandsraum beschrieben sein.

3.2 ☞ Erstellen Sie das Modell **modell3=tf(1, [0.025, 0.12, 1])**. Wandeln Sie es in den Zustandsraum (**modell3ss**) um. Wie viele Zustände besitzt es? Warum ist die Matrix **D** Null?

Der Anfangszustand von modell3 sei [0, 1]. Stellen Sie den Verlauf des Ausgangssignals durch den Befehl **initial(modell3ss, [0, 1])** dar. Stellen Sie anschließend mit Hilfe von **ltiview** die Gewichtsfunktion dar. Vergleichen Sie die beiden Darstellungen.

Wandeln Sie modell3 in ein zeitdiskretes Modell **modell3d** mit der Abtastzeit von 0.07 um. Plotten Sie die Übergangsfunktion.

4 Einführung in Simulink

4.1 Erste Schritte und Bedienungshilfen...

Einstieg in Simulink

SIMULINK ist ein Programm zur Lösung linearer und nichtlinearer Differentialgleichungen, die das Verhalten physikalischer dynamischer Systeme durch ihre mathematischen Modelle beschreiben. Dazu besitzt SIMULINK eine graphische und blockorientierte Oberfläche, mit deren Hilfe die Gleichungen in Form von (Übertragungs-)Blöcken wie bei einem Wirkungsplan eingegeben und dargestellt werden.

Eine große Anzahl an vordefinierten Blöcken sind in sogenannten Bibliotheken zusammengefasst. Mit Hilfe der Maus können die Blöcke auf die Arbeitsfläche gezogen und anschließend parametriert werden.

Zuerst muß SIMULINK gestartet werden. Dies geschieht entweder durch Eingabe von **simulink** in dem MATLAB-Befehlsfenster oder durch Drücken des SIMULINK-Symbols in der Symbolleiste von Matlab. Es öffnet sich der SIMULINK Library Browser, der den Zugriff auf die verschiedenen „Bibliotheken“ erlaubt. Die Bibliotheken sind in verschiedene Gruppen unterteilt: Continuous, Discrete, ...

Durch Drücken des Symbols **Create new model** öffnet sich ein leeres Fenster ohne Namen (**untitled**). Der Name wird über **File** → **Save** beim Abspeichern eingegeben. SIMULINK-Modelle haben automatisch die Endung **.mdl**. Die mdl-Datei eines Modells dient zur Speicherung der Modellbestandteile, der Signalverbindungen und der Simulationsparameter. Eine mdl-Datei ist eine Textdatei und kann mit dem Editor verändert werden.

Geben Sie nun Ihrem neuen Modell den Namen **test1**.

Geöffnet wird eine Blockbibliothek durch einen Doppelklick auf ihr Symbol oder durch einen einfachen Klick auf das +. Mit der gedrückten linken Maustaste kann dann eine Kopie des gewünschten Blocks auf das Modellfenster gezogen werden (Klicken & Ziehen). Sie können im Modellfenster mehrere Blöcke markieren, indem Sie beim Anklicken der einzelnen Blöcke die **Shift**-Taste gedrückt halten.

Innerhalb des Modellfensters können

- Blöcke durch Klicken & Ziehen mit der **linken** Maustaste **verschoben** werden,
- Blöcke durch Klicken & Ziehen mit der **rechten** Maustaste **kopiert** werden,
- Blöcke mit Hilfe der Menüpunkte **Format** → **Flip Block** oder **Format** → **Rotate Block** gedreht bzw. gespiegelt werden,
- **Signalverbindungen** durch Klicken & Ziehen mit der linken Maustaste von Blockausgängen zu Blockeingängen erzeugt werden,
- **Signalabzweigungen** durch Klicken & Ziehen mit der **rechten** Maustaste auf eine Signallinie erzeugt werden.

Blockbibliotheken

Folgende Block-Bibliotheken sind im Rahmen dieser Kurzanleitung von besonderem Interesse:

- Continuous
  enthält Elemente zur Darstellung linearer kontinuierlicher Systeme in Form der Übertragungsfunktion und im Zustandsraum.
- Discontinuities
  enthält nichtlineare Elemente zur Modellierung von Sättigungen (Saturation), Quantisierungen, Totzonen, Hysteresen etc.
- Discrete
  enthält Elemente zur Darstellung linearer zeitdiskreter Systeme in Form der Übertragungsfunktion und im Zustandsraum. Jeder diskrete SIMULINK-Block ist mit einem Abtaster und Halteglied 0. Ordnung ausgestattet.
- Math Operations
  enthält Basiselemente mathematischer Operationen (+, -, $*$, /, min/max, abs, sin, ...). Häufig benötigte Elemente sind **sum** und **gain** (Konstanter Faktor). Auch logische Operationen sind hier vorhanden.
- Signal Routing
  enthält Blöcke zum Zusammenfassen und Trennen von Signalen (z. B. Mux, Demux), Blöcke zur Erzeugung und Auswertung von Signalbussen, Schalter (Switches), Goto- und From-Blöcke etc.
- Sinks
  enthält Elemente zur Ausgabe, zum Darstellen und zum Speichern von Signalen. Z. B. können mit dem Block To Workspace Simulationsergebnisse in den MATLAB-Workspace geschrieben werden.
- Sources
  enthält Blöcke mit möglichen Eingangssignalen. Neben vordefinierten Signalformen (z. B. Step (Sprungfunktion), Constant) können u. a. auch Variablen aus dem Workspace mit beliebigen in Matlab erzeugten Werten als Eingangssignal angegeben werden (From Workspace).
- User-Defined Functions
  In dieser Bibliothek liegen Blöcke, mit denen eigene Funktionen realisiert werden können. (siehe auch Abschnitt 4.4).

Bearbeiten von Blöcken und Signalen

Meistens benötigt man andere Parameter als diejenigen, die standardmäßig bei den SIMULINK-Blöcken eingetragen sind. Zum Ändern der Parameter wird ein Blöck mit einem Doppelklick geöffnet. Hier erhält man auch die **Hilfe** (html) zu einzelnen Blöcken. Als Parameter können MATLAB-Variablen aus dem Workspace eingegeben werden. Sinnvollerweise speichert man alle in einem Modell benötigten Variablen entweder in einem mat-File, das vor der Simulation geladen wird oder erzeugt Sie durch Ausführen eines m-Files, das die entsprechenden Definitions-Befehle enthält.

Durch einen Doppelklick auf Signale kann diesen ein Name gegeben werden. Der Name eines Blocks wird durch einfaches Anklicken (des Namens) und Editieren geändert. Das Menü Format ermöglicht, die Anzeige des Namens ein- und auszuschalten. Darüber hinaus können weitere Formate wie Zeichenfont, Farben, Schatten... eingestellt werden.

Aufgaben und Übungen

4.1 ☞

- Kopieren Sie einen Block **Transfer Fcn** aus der **Continuous**-Bibliothek. Speichern Sie das Modell als **test1.mdl** und schließen sie es. Sie können es durch Doppelklick auf die entsprechende Datei im Fenster **Current Directory** wieder laden.

  Oder Sie geben den Befehl **test1** im MATLAB-Befehlsfenster ein. Geben Sie als Eingangssignal auf die Übertragungsfunktion einen Sprung (engl. *step*) und stellen Sie das Ausgangssignal in einem Scope dar. Starten Sie die Simulation durch Drücken des Startsymbols ▷ in der Symbolleiste oder im Menü **Simulation** → **Start**und überprüfen Sie das Ergebnis. Speichern Sie das Modell.

- Geben Sie nun als Eingangssignal u eine Sinusschwingung $u = \sin(4t) + 2$ auf das System und sehen Sie sich das Ausgangssignal in einem Scope an.

- Stellen Sie mit Hilfe des **Mux**-Blocks Ein- und Ausgangssignal im gleichen Scope dar und ändern sie den Übertragungsfaktor der Übertragungsfunktion auf 3. Speichern Sie anschließend das Modell unter dem Namen **test2**.

- Ändern Sie das Eingangssignal zu einem Puls der Amplitude 3, der Periode 3s und der Breite von 1s und simulieren Sie das Ergebnis.

4.2 Block-Bibliothek Sinks

Scopes

Scopes dienen dem Beobachten und evtl. Speichern von Signalen. Öffnen Sie zuerst das Modell test2, welches ein Scope enthält. Simulieren Sie das Modell und öffnen Sie anschließend das Scope durch einen Doppelklick. Standardmäßig enthält die Zeitachse (X-Achse) die Simulationsdauer und die Y-Achse den Bereich von -5 bis 5. Durch die Symbolleiste können die Einstellungen des Scopes verändert werden.

Die Symbolleiste (siehe Abb. 4.1) enthält die folgenden Funktionen von links nach rechts:

- Drucken
- Parameter. Dieses Symbol öffnet ein neues Dialogfenster.
  - In der Karte **General** können folgende Einstellungen getroffen werden:
    - **Number of axes** ändert die Anzahl der Subplots. Die einzelnen Subplots besitzen die gleiche Zeitachse.
    - **floating scope** erlaubt die Darstellung von Signalen, ohne sie mit dem Scope zu verbinden. Details können der MATLAB-Hilfe entnommen werden.

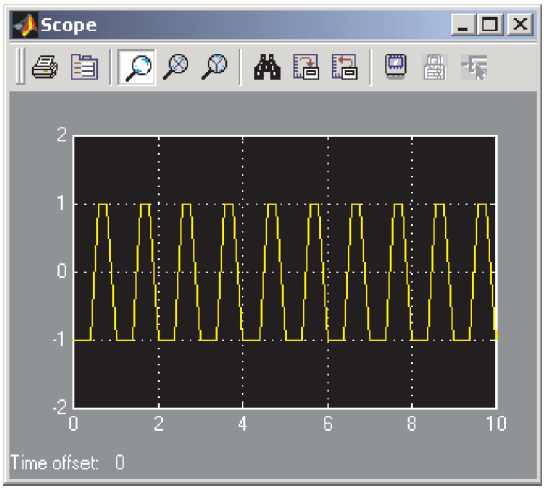


Abbildung 4.1: SIMULINK-Scope

  - ∗ **Time Range** ermöglicht, andere Zeitbereiche als die Simulationsdauer (auto) einzustellen.
  - ∗ **Tick Labels** können hier ein- und ausgeschaltet werden.
  - ∗ **Sampling** stellt mit Decimation ein, jeder wievielte Wert im Scope dargestellt werden soll, wobei in dem Fall die Zeitintervalle nicht konstant sind. Mit Sample Time wird die Länge konstanter Zeitintervalle festgelegt.
- Die Karte **Data History** erlaubt das Abspeichern der Simulationsergebnisse in einer Variablen (**Save Data to Workspace**, Standardvariable: ScopeData) mit dem eingestellten Typ: Struktur oder Array. Es wird sowohl die Zeit als auch die Daten gespeichert. Standardmäßig ist die Anzahl der im Scope dargestellten und zu speichernden Punkte auf 5000 begrenzt (**Limit data points to last**). Die Grenze kann verändert oder durch das Löschen der Markierung ganz aufgehoben werden.

- Zoomen in X- und Y-Richtung. Der zu vergrößernde Bereich wird durch das Ziehen mit gedrückter linker Maus-Taste markiert. Durch Drücken der rechten Maustaste wird die Vergrößerung wieder rückgängig gemacht.
- Zoomen in X-Richtung, Vergrößern und Verkleinern wie oben.
- Zoomen in Y-Richtung, Vergrößern und Verkleinern wie oben.
- Autoscale. Stellt die X- und Y-Achse so ein, dass der simulierte Graph gerade ganz sichtbar ist.
- Save current axes settings. Dieses Symbol ermöglicht das Speichern der aktuellen Achseneinstellungen.
- Restore saved axes settings: Zurückholen der gespeicherten Achseneinstellungen.
- Die letzten drei Symbole betreffen die FLOATING SCOPES. Details entnehmen Sie bitte der Hilfefunktion.

Andere Senken

Zwei weitere Möglichkeiten zum Abspeichern von Signalen sind mit den Blöcken **To Workspace** und **To File** vorhanden. Das Abspeichern mit To Workspace ist identisch zum Scope mit den gleichen Einstellungen. To File verlangt nach dem Datei- und Variablennamen, der Abtastzeit und speichert die Variable als Matrix im angegebenen File. Die Zeit wird jeweils mitgespeichert.

Display zeigt den aktuellen Wert des Signals numerisch an. XYGraph trägt ein Signal über einem anderen auf. Stop Simulation beendet die Simulation, sobald das Eingangssignal ungleich Null wird.

4.3 Simulationsparameter

Die Simulationsparameter werden im Menü **Simulation → Configuration Parameters...** eingestellt. Es erscheinen unter anderem die Karten **Solver**, **Data Import/Export**und **Diagnostics**.

Solver

Die **Simulation time** legt fest, für welchen Zeitraum das Modell simuliert werden soll. Dies geschieht nicht in Echtzeit, sondern hängt davon ab, wie lange der Rechner für die Berechnungen benötigt. Die Startzeit kann dabei sowohl kleiner, gleich als auch größer Null sein. Wird an irgendeiner Stelle im Modell ein Startwert (**Initial Condition**) angegeben, so gilt diese für den festgelegten Startzeitpunkt der Simulation und nicht automatisch für Null. Die **Simulation stop time** kann entweder in dem entsprechenden Feld bei **Simulation time** eingeben werden, oder direkt in der Toolbar, rechts neben dem Stop-Zeichen.

Simulieren bedeutet, dass das in Blöcken erstellte Differentialgleichungssystem mit Hilfe numerischer Integrationsverfahren schrittweise gelöst wird. Bei den **Solver options** wird der gewünschte Integrationsalgorithmus ausgewählt, festgelegt, ob mit oder ohne Schrittweitensteuerung gearbeitet wird (**Fixed-step** oder **Variable-step**), und Werte für die maximale Schrittweite (**Max step size**), die Anfangsschrittweite (**Initial step size**) sowie die Fehlertoleranzen der Schrittweitensteuerung festgelegt.

Es wird empfohlen, als Integrationsalgorithmus für kontinuierliche Systeme **ode45** und für diskrete Systeme **discrete** einzustellen. Die anderen Möglichkeiten erfordern eine bessere Kenntnis der numerischen Mathematik.

Data Import/Export

Anstatt mit dem (Source-)Block **Load from workspace** kann auf Signale aus dem Workspace auch mit dem Block **In1** (Input Port) aus der **Signals & Systems** Bibliothek zugegriffen werden, sobald ihre Variablennamen im Feld **Load from workspace** eingegeben worden sind. Die erste Spalte der eingegebenen Matrix wird als Zeitvektor genommen.

Save to workspace funktioniert analog mit dem **Out1** (Output Port) und stellt daher eine weitere Möglichkeit dar, Simulationsdaten im MATLAB-Workspace anzulegen. Die **Save options** sind ebenfalls analog zu den Einstellungen in den Scopes und den To Workspace-Blöcken.

Mit den **Output options** (nur verfügbar bei variabler Schrittweite, siehe Solver) können zusätz-

liche Punkte in einem Integrationsintervall der Ausgabe hinzugefügt werden. Der **Refine factor** gibt an, in wie viele kleinere Intervalle ein Integrationsintervall mit Hilfe der Interpolation für die Ausgabe aufgeteilt werden soll. **Produce additional output** zwingt den Integrationsalgorithmus Schritte zu den angegebenen Zeiten zusätzlich zu berechnen. Im Gegensatz dazu wird mit **Produce specified output only** nur zu den angegebenen Zeitpunkten (mit Start- und Endzeit) ein Integrationsschritt durchgeführt.

Diagnostics

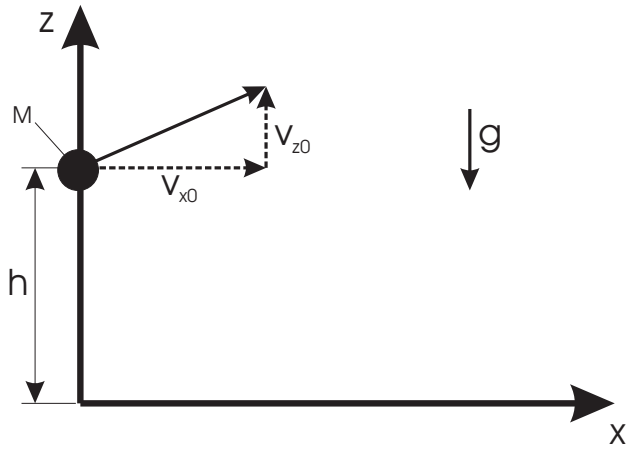
Auf dieser Karte kann man auswählen, bei welchen Ereignissen, welche Art von Fehlermeldungen ausgegeben werden sollen: keine (**none**), Warnung (**warning**) oder Fehler (**error**). Im Allgemeinen sollten die Standardeinstellungen bis auf wenige Ausnahmen nicht verändert werden müssen.

Eine sehr wichtige Meldung ist diejenige eines **Algebraic loop**. Eine sogenannte algebraische Schleife tritt dann auf, wenn ausschließlich Blöcke mit Durchgriff einen geschlossenen Wirkungskreis bilden. Bei Blöcken mit Durchgriff hängt das aktuelle Ausgangssignal vom aktuellen Eingassignal ohne Verzögerung ab, z. B. beim Gain. Die Rechenzeit wird durch eine algebraische Schleife stark verlangsamt. Durch Einfügen eines **Memory**-Block aus der Continuous-Bibliothek lässt sich die Schleife aufbrechen. Eine weitere Möglichkeit liegt in der Verwendung des **Algebraic Constraint**-Blocks aus der Math Operations-Bibliothek zur Lösung algebraischer Gleichungs-/Differentialgleichungssysteme.

Aufgaben und Übungen

4.2 ☞ „Schiefer Wurf“

Simulieren Sie die Flugbahn einer punktförmigen Masse M in der (x, z)-Ebene, welche im Punkt $(0, h)^T$ mit der Geschwindigkeit $(v_{x0}, v_{z0})^T$ losgeworfen wird:



Vernachlässigen Sie zunächst alle Reibungs- und Kontakteffekte.
Es gilt: $M = 2\text{kg}$, $v_{x0} = v_{z0} = 10\frac{\text{m}}{\text{s}}$, $h = 5\text{m}$, $g = 9.81\frac{\text{m}}{\text{s}^2}$.
Tipp: Es gilt

$$\begin{aligned} \dot{x} &= v_{x0} \\ \ddot{z} &= -g \end{aligned}$$

- Setzen Sie diese beide Gleichungen mit Hilfe von Integrierern in Simulink um. Vergessen Sie nicht die Anfangsbedingungen. Speichern Sie das System unter dem Namen **schiefer-Wurf.mdl**.

- Simulieren Sie das System für eine Zeitdauer von $t_{sim} = 5\text{s}$. Stellen Sie die Zeitverläufe von $x(t)$, $\dot{x}(t)$, $z(t)$ und $\dot{z}(t)$ in einem Scope und $x(t)$ und $z(t)$ in einem x-y-Plot dar.

In diesem einfachsten Fall ist neben der numerischen Lösung des Differentialgleichungssystems (Simulink) natürlich auch eine analytische Lösung möglich.

- Integrieren Sie das System per Hand und ermitteln Sie die analytische Lösung. Stellen Sie die Lösungen für $x(t)$ und $z(t)$ in einem weiteren Scope dar und vergleichen Sie die beiden Lösungen.
  Tipp: Sie benötigen den Clock-Block aus der Sources-Bibliothek.

4.3 ☞

Im folgenden sollen Reibungs- und Kontakteffekte in die Simulation **schieferWurf** integriert werden, so dass die Lösung der DGL nunmehr numerisch erfolgen kann.

- Ergänzen Sie die Simulation um einen „Boden“ $z = 0$. Modellieren Sie ihn in der Weise, dass er sich in z-Richtung wie ein Feder-Dämpfer-Glied verhält. Kommt also die Punktmasse mit dem Boden in Kontakt, überträgt dieser folgende Kraft auf die Punktmasse (in z-Richtung):

  $$F_{zB} = -k_b z - k_d \dot{z} \quad \text{für } z < 0$$

  mit $k_b = 1\frac{\text{kN}}{\text{m}}$, $k_d = 10\frac{\text{Ns}}{\text{m}}$.
  Simulieren Sie nun für eine Zeitdauer von $t_{sim} = 10\text{s}$.

- Ergänzen Sie nun den Bodenkontakt um eine Reibkomponente in x-Richtung. Im Kontaktfall wird gemäß dem Reibungsgesetz in x-Richtung folgender Betrag der Reibkraft auf den Massepunkt übertragen:

  $$|F_R| = \mu |F_N| \quad \text{für } z < 0,$$

  wobei $\mu = 0.1$. F_N ist die Normalkraft, mit welcher der Massepunkt auf den Boden drückt. Überlegen Sie sich selbst die Richtung der Kraftwirkung.

- Ergänzen Sie das Modell um Luftreibung. Der Betrag der Luftreibungskraft ergibt sich zu

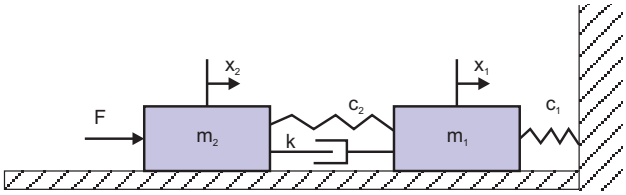
  $$|\vec{F}_L| = c_L |\vec{v}|^2 = c_L \left| \begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} \right|^2,$$

  wobei $c_L = 0.01\frac{\text{Ns}^2}{\text{m}^2}$. Ihre Richtung weist entgegen dem Geschwindigkeitsvektor $\vec{v} = \begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix}$.

Wie weit kommt der Massepunkt in x-Richtung in Ihrer Simulation? In welcher z-Position kommt er zu Ruhe? Ab welchem Zeitpunkt kommt die Simulation nicht mehr weiter? Weshalb? Wie kann man dieses Problem umgehen?

4.4 ☞ Laden Sie das Modell **schieferWurf**. Speichern Sie den simulierten Verlauf von $x(t)$ als Matrix im Workspace unter dem Variablennamen **xt**. Speichern Sie den Verlauf von $z(t)$ in einer Datei **testdat.mat** unter dem Variablennamen **zt**. Kontrollieren Sie das Ergebnis im Workspace und in der Datei. Ändern Sie den Decimation Factor auf 10 und 100 und betrachten Sie das Ergebnis.

4.5 ☞ Gegeben ist folgendes mechanische System mit $m_1 = 1, m_2 = 5, c_1 = 3, c_2 = 1, k = 0.5$:

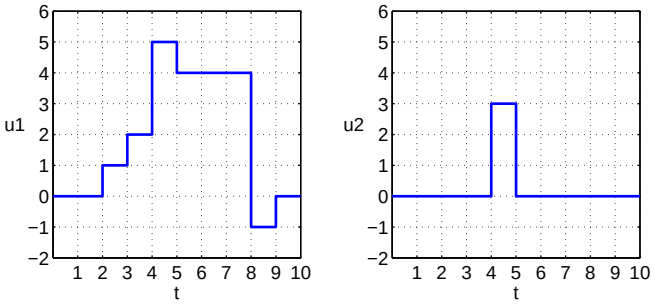


- Stellen Sie die Differentialgleichungen der Bewegung der beiden Massen mit dem Eingangssignal F auf.
- Erstellen Sie ein Blockschaltbild, das neben Elementen aus den Blockbibliotheken Quellen und Senken ausschließlich die Blöcke vom Typ **Gain**, **Integrator** und **Sum** enthält. Speichern Sie das Modell als **Test3**.
- Simulieren sie nun das System über eine Zeitdauer von 30 Sekunden, falls ein Einheitskraftsprung auf das System aufgebracht wird. Sind die Ergebnisse plausibel?
- Ändern Sie nun den Refine Factor auf 10 und vergleichen Sie die Simulationsergebnisse. Ändern Sie die Fehlertoleranzen sowohl zu genaueren als auch ungenaueren Werten. Vergleichen Sie die Simulationsergebnisse.

4.6 ☞ Simulieren Sie folgendes Differentialgleichungssystem mit einem Zustandsraummodell:

$$\dot{\mathbf{x}} = \begin{bmatrix} -0.2667 & -0.3333 \\ 1 & 0 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 1 & -2 \\ 0 & 0.5 \end{bmatrix} \mathbf{u}, \quad \mathbf{y} = \begin{bmatrix} 1.67 & 0 \\ 0 & 1.2 \\ 1 & 1 \end{bmatrix} \mathbf{x} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \mathbf{u}$$

Simulieren Sie das System mit ode45 und variabler Schrittweite für die Dauer von 20 Sekunden, während am Eingang folgende Signale anliegen. Geben Sie **y** auf einem Scope aus.



Tip: Definieren Sie im MATLAB-Workspace die Vektoren

```
u1 = [0 0 1 2 5 4 4 4 -1 0 0]'
```

```
u2 = [0 0 0 0 3 0 0 0 0 0 0]'
t = [0:10]'
```
und benutzen Sie in Simulink den „From-Workspace“-Block.

4.7 ☞ Erstellen Sie ein zeitdiskretes Zustandsraummodell mit den selben Matrizen und Eingangssignalen wie in der vorigen Aufgabe. Simulieren sie mit fester Integrationsschrittweite von 0.1 und diskretem Integrationsalgorithmus. Simulieren Sie anschließend mit der Integrationsschrittweite 0.02 und 2. Vergleichen Sie die Ergebnisse.

4.4 Selbstdefinierte Strukturen

Subsystems

Subsystems bieten die Möglichkeit der Hierarchisierung (mit beliebig vielen Ebenen), die für größere Modelle unumgänglich ist. Es gibt zwei Möglichkeiten, Subsysteme zu erzeugen: zum einen über den Block **SubSystem** aus der **Ports & Subsystems**-Bibliothek, zum anderen im Modellfenster durch vorheriges Markieren mit dem Markierungsfenster (nicht nur durch Anklicken der Blöcke!) der in ein Untersystem zu verschiebenden Blöcke und anschließender Auswahl des Menüpunkts **Edit → Create Subsystem**.

Durch Doppelklicken werden Subsysteme innerhalb eines übergeordneten Modells geöffnet. Sollen Signale an ein Subsystem gegeben oder von dem Subsystem erhalten werden, muss das Subsystem Inport bzw. Outport-Blöcke enthalten. Beim Zusammenfassen über das Markieren werden sie automatisch erzeugt.

Erstellen Sie im Modell `test2` ein Subsystem, welches die Blöcke `Sine Wave` und `transfer function` enthält. Ändern sie in den Einstellungen von `Sine Wave` den Parameter `Frequency` auf die Variable **freq**.

Noch ein Hinweis: Die Blöcke **Enable** und **Trigger** aus der **Ports & Subsystems**-Bibliothek ermöglichen die bedingte Ausführung eines Subsystems. An dieser Stelle soll darauf nicht weiter eingegangen werden.

Maskierte Blöcke

Mit Hilfe des Menüs **Edit → Mask Subsystem** können Subsysteme maskiert werden, d. h. ihnen auf der übergeordneten Hierarchie ein bestimmtes Aussehen und Funktion gegeben sowie ihre Inhalte versteckt werden.

Führen Sie den Menübefehl mit dem in `test2` erstellten Subsystem durch. Es erscheint das Fenster **Mask Editor : SubSystem** mit den vier Karten **Icon, Parameters, Initialization** und **Documentation**.

- Icon
  erlaubt, ein eigenes Symbol für den Block festzulegen. Bei den **Drawing Commands** sind alle Befehle zur Darstellung des neuen Symbols anzugeben. Text kann mit **disp**, Funktionsverläufe mit **plot** und Grafiken mit **image** gezeichnet werden.

  Geben Sie folgende Befehle ein:

```
disp('Testsystem\n'); plot([0:0.01:0.9],sin(2*pi*[0:0.01:0.9]))
```

- Parameters
  Die im Innern des Subsystems verwendeten Variablen (hier: `freq`) müssen in der Variablenliste festgelegt werden. Ihre Namen werden im Feld **Variable** eingetragen, dazu passende Erläuterungen bei **Prompt**. Das Feld **Type** legt den Typ des Eingabefeldes fest (Editierfeld, Checkbox, Auswahlliste). Ist **Evaluate** markiert, wird der einzugebende Parameter von Matlab ausgewertet und falls möglich als Zahl übergeben, sonst als String. Ist **Tunable** markiert, können die Parameter zur Simulationslaufzeit verändert werden.

  Werden Variablen in dieser Liste als Parameter festgelegt, erscheint bei Doppelklick auf das Subystem für jeden Parameter das festgelegte Eingabefeld.

  Drücken Sie den Knopf **Add**. Tragen Sie dann bei Variable `freq` und bei Prompt `Sinus-Frequenz` ein.

- Initialization
  erlaubt die Definition von lokalen Variablen und nötigen Initialisierungsberechnungen. Bei den **Initialization Commands** sind alle MATLAB-Befehle zulässig.

- Documentation
  erlaubt die Funktionalitäten des maskierten Blocks zu beschreiben und einen Hilfe-Text für diesen Block zu erstellen.

  Bei **Mask Type** wird der Maske ein Name gegeben. Tragen Sie hier „Mein Block“ ein. Geben Sie bei **Mask Description** „Dieser Block hilft, das Maskieren von Blöcken zu verstehen.“ ein. Dieser Satz erscheint immer in der neuen Blockmaske. Geben Sie bei **Mask Help** den Text „Hier erscheint der Hilfe-Text“ ein. Die Hilfe wird automatisch im HTML-Format erstellt, so dass auch HTML-Befehle eingegeben werden können.

Verlassen Sie den Mask Editor mit **OK** und probieren Sie das Ergebnis aus. Sie können die Simulation starten, sobald Sie einen Wert für `freq` eingegeben haben.

Hinweis: Eine Reihe von den SIMULINK Standardblöcken sind maskierte Subsysteme.

S-Functions

S-Functions sind MATLAB-Funktionen (oder C oder Fortran, was hier nicht weiter behandelt werden soll, s. Kap. 2) mit festgelegter Parameterstruktur, mit deren Hilfe beliebige lineare und nichtlineare Differentialgleichungs-Systeme in Zustandsraumdarstellung modelliert werden können.

Der Aufruf von SIMULINK erfolgt über die Verwendung des Blocks **S-Function** aus der Bibliothek **User-Defined Functions**. In der Eingabemaske wird der Funktionsname der zu verwendenden S-Function sowie bei Bedarf die zu übergebenden Parameter eingegeben. Eine S-Function kann maskiert werden, sodass Erläuterungen zu den eingegebenen Parametern und der Wirkungsweise des Blockes hinzugefügt werden können.

Der Funktionskopf einer S-Function besitzt folgendes Aussehen:

```
[sys, x0, str, ts] = sfunctionname(t, x, u, flag);
```

oder bei Eingabe von zusätzlichen Parametern p1 bis pn:

```
[sys, x0, str, ts] = sfunctionname(t, x, u, flag, p1, ... , pn);
```

Mit dem Parameter **flag** wird bestimmt, was die S-Funktion beim aktuellen Aufruf aus der Zeit t, dem aktuellen Zustand x und dem aktuellen Eingangssignal u berechnet. Die Steuerung des Aufrufs und damit des Parameters flag liegt vollständig intern bei SIMULINK.

- flag = 0: Initialisierung
  Erfolgt einmalig zu Simulationsbeginn. Die Variable sys muss ein Vektor mit 7 ganzzzahligen Werten sein, wobei die Werte z. B. die Anzahl an Ein- und Ausgängen und Zuständen festlegen.

- flag = 1: Berechnung der Ableitung kontinuierlicher Zustandsgrößen
  Erfolgt bei jedem Simulationsschritt. Hier wird die kontinuierliche Zustands-Differentialgleichung

  sys = f(x, u) (% $=\dot{x}$)

  programmiert. sys ist jetzt ein Vektor mit so vielen Einträgen wie das System kontinuierliche Zustände besitzt. Dies wurde bei der Initialisierung festgelegt.

- flag = 2: Aktualisierung diskreter Zustände
  Erfolgt bei jedem Simulationsschritt. Hier wird die diskrete Zustands-Differenzengleichung

  sys = f(x_k, u_k) (% $=x_{k+1}$)

  programmiert. sys ist jetzt ein Vektor mit so vielen Einträgen wie das System diskrete Zustände besitzt. Dies wurde bei der Initialisierung festgelegt.

- flag = 3: Berechnung der Ausgangsgrößen
  Erfolgt bei jedem Simulationsschritt. Hier wird die Ausgangsgleichung

  sys = g(x, u) (% = y)

  programmiert. sys ist jetzt ein Vektor mit so vielen Einträgen wie das System Ausgänge besitzt. Dies wurde bei der Initialisierung festgelegt.

- flag = 9: Abschließende Tasks
  Erfolgt einmal zu Simulationsende. Hier werden alle abschließenden Befehle eingegeben, z. B. Variablen aus dem Workspace löschen.

Eine ausführliche Erläuterung finden Sie in der Datei **sfuntmpl.m**, die als Vorlage für selbsterstellte S-Functions dient. Eine einfache S-Function ist die Datei **sfuncont.m**, die einen Integrierer darstellt. Beide Dateien sind im Verzeichnis <MATLAB> \toolbox\simulink\blocks enthalten.

Erstellen einer eigenen Bibliothek

SIMULINK bietet die Möglichkeit, eigene Bibliotheken mit z. B. häufig benutzten Blöcken zu erstellen. Dazu wird ein neues Fenster über das Menü **File → New → Library** als Bibliothek und nicht wie sonst als Modell erstellt.

Alle in dieses Fenster kopierten Blöcke sind nach dem Abspeichern in Form einer neuen Bibliothek, die den Namen der Datei erhält, enthalten. Die Bibliothek kann später beliebig verändert und erweitert werden. Nach dem Abspeichern und Schließen ist die Bibliothek automatisch gesperrt und muss für gewünschte Veränderungen erst durch den Menü-Befehl **Edit → Unlock Library** freigegeben werden. Werden in Bibliotheken Änderungen an Blöcken vorgenommen, aktualisieren sich automatisch alle Modelle, in denen diese Blöcke vorkommen.

Durch Kopieren aus dem Bibliotheksfenster in ein Modellfenster werden Kopien der Bibliotheks-Blöcke erstellt. Die Verbindung wird dabei über den Blocknamen hergestellt, sodass dieser in der Bibliothek nicht mehr verändert werden sollte.

Aufgaben und Übungen

4.8 ☞ Erstellen Sie aus einem Subsystem, das die Übertragungsfunktion eines PT1 enthält, einen maskierten Block mit dem Maskennamen PT1. Der Übertragungsfaktor und die Zeitkonstante sollen als Parameter eingegeben werden können. Das Symbol des Blocks soll die Übergangsfunktion enthalten. Geben Sie sinnvolle Erläuterungs- und Hilfetexte. Simulieren Sie mit diesem Block für verschiedene Kennwerte die Übergangsfunktion und die Antwort auf eine Sinus-Funktion.

4.9 ☞ Öffnen Sie „sfuncont.m“ (mit dem Befehl `open sfuncont`), speichern Sie die Datei als „doppint.m“ in Ihrem Arbeitsverzeichnis, und simulieren Sie die Übergangsfunktion mit Hilfe des Blocks „s-function“. Ändern Sie anschließend die Funktion in einen Doppelintegrierer ($G(s) = \frac{1}{s^2}$).

Ändern Sie die Funktion von doppint, sodass ein Übertragungsfaktor als zusätzlicher Parameter angegeben wird. Maskieren Sie den Block in SIMULINK, und geben Sie den Parameter als Variable mit dem Erklärungstext „Übertragungsfaktor “an.

4.10 ☞ Programmieren Sie eine S-Function zur Simulation eines PT1-Elements.

4.11 ☞ Erzeugen Sie eine Bibliothek mit dem PT1-Element als Inhalt.

5 Einführung in Stateflow

Das Paket STATEFLOW® dient zur Modellierung und Simulation ereignisdiskreter Systeme innerhalb von SIMULINK®. Mit Hilfe sogenannter *Zustandsdiagramme* (engl. *state charts*) werden die Modelle graphisch programmiert. Eine Hierarchiebildung ermöglicht eine Aufteilung, welche oft in enger Anlehnung zum realen Prozess gebildet werden kann. Durch eine Übersetzung in eine s-function wird der Rechenzeitbedarf während der Ausführung in Grenzen gehalten.

Es empfiehlt sich, gerade bei umfangreicheren diskreten (Teil-) Systemen, aber auch schon bei relativ kleinen Systemen mit wenigen Zuständen, STATEFLOW® einzusetzen, statt diese Funktionen umständlich mit SIMULINK®-Blöcken zu realisieren. Zu beachten ist allerdings, dass STATEFLOW® ein sehr mächtiges Werkzeug ist: Ein und dasselbe System lässt sich in STATEFLOW® auf sehr unterschiedliche Arten realisieren, wobei selbst darauf zu achten ist, dass die gewählte Realisierung möglichst übersichtlich und nachvollziehbar ist.

5.1 Grundelemente von Stateflow

Die Modellierung der ereignisdiskreten Systeme erfolgt in STATEFLOW® mit *Zustandsübergangsdiagrammen* (engl. *state charts*), welche mit einem graphischen Editor erzeugt werden. Die Charts müssen in ein SIMULINK®-Modell eingebettet sein, welches den Aufruf und den Ablauf des State Charts steuert.

Mit der Befehlseingabe von

```
≫ stateflow
```

oder

```
≫ sf
```

in der MATLAB-Kommandozeile startet man ein leeres Chart innerhalb eines neuen SIMULINK®-Modells. Alternativ kann man mit Hilfe des Simulink-Library-Browsers aus der Toolbox „Stateflow“ ein leeres Chart durch Ziehen mit der Maus in ein bestehendes SIMULINK®-Modell einfügen.

Hinweis: Zur Ausführung eines Charts wird zwingend ein C-Compiler benötigt. Eventuell muss dieser in MATLAB zuerst mit dem Befehl

```
≫ mex -setup
```

konfiguriert werden.

Die Bedienoberfläche

Durch Doppelklick auf das Chart-Symbol im SIMULINK®-Modell wird der graphische Editor von STATEFLOW® gestartet (siehe Abb. 5.1). Mit der Werkzeugleiste auf der linken Seite können die verschiedenen Chart-Elemente wie Zustände (States) und Transitionen ausgewählt und mit der Maus auf der Arbeitsfläche platziert werden. Kommentare können durch Doppelklicken auf eine leere Fläche eingefügt werden.

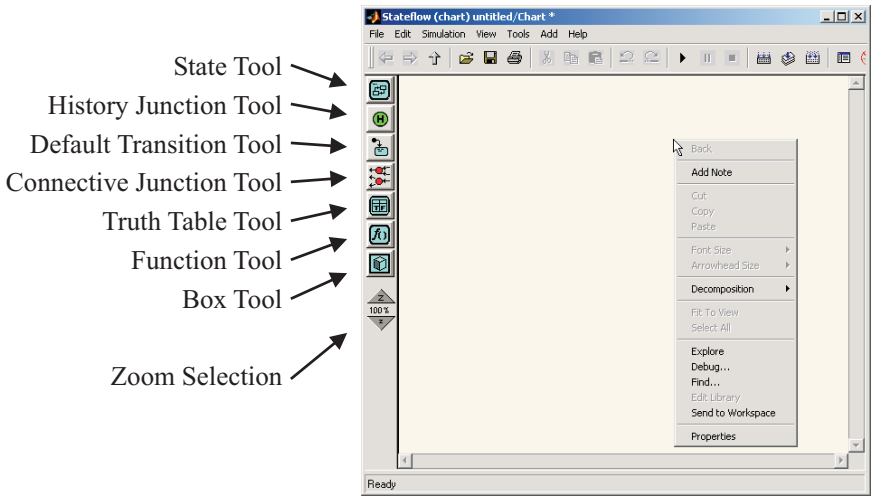


Abbildung 5.1: Graphischer Stateflow Editor

Der Zustand

Nach Auswahl des State-Tools in der Werkzeugleiste kann durch Klick ein Zustand auf der Arbeitsfläche platziert werden (Abb. 5.2). Zustände können verschoben werden, indem in den leeren Raum im Inneren des Zustandes geklickt und gezogen wird. Innerhalb des Zustandes steht sein Label als Fließtext, welches zwingend vergeben werden muss. Ist noch kein Label eingegeben, erscheint stattdessen ein Fragezeichen.

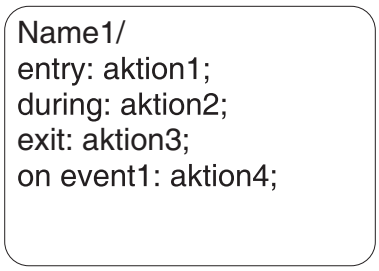


Abbildung 5.2: Zustand mit komplettem Label

Zum Bearbeiten des Labels klickt man bei markiertem Zustand auf das Fragezeichen bzw. auf das schon vorhandene Label. Das Label (vgl. Abb. 5.2) besteht dabei aus `Zustand1`, dem Namen des Zustandes, welcher eindeutig (innerhalb einer Hierarchieebene, siehe Abschnitt 5.2) als gültiger C-Variablenname vergeben werden muss. Die weiteren Elemente des Labels sind optional und bezeichnen Aktionen, welche durch diesen Zustand angestoßen werden. Die Syntax für die Festlegung der Aktionen ist die *Action Language*, eine Mischung aus C und der aus MATLAB bekannten Syntax (siehe Abschnitt 5.3). Folgende Schlüsselwörter legen fest, dass die Aktion ausgeführt wird,

entry: wenn der Zustand aktiviert wird,

during: wenn der Zustand aktiv ist (und das Chart ausgeführt wird),

exit: wenn der Zustand verlassen wird,

on event: wenn der Zustand aktiv ist und das angegebene Ereignis auftritt.

Die Transition

Eine Transition für einen Zustandsübergang erzeugt man, indem man auf den Rand des Augangszustandes klickt und die Maus bis zum Rand des folgenden Zustandes zieht und dort loslässt. Dabei entsteht ein die Transition symbolisierender Pfeil, dessen Form und Lage durch Ziehen verändert werden kann.

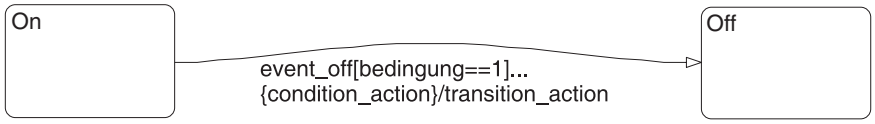


Abbildung 5.3: Transition mit komplettem Label

Ebenso wie der Zustand besitzt die Transition ein Label, welches hier aber nicht zwingend vergeben werden muss. Ein Transitionslabel besitzt die Syntax

```
event[condition]{conditionAction}/transitionAction
```

Alle Komponenten des Labels sind optional und können beliebig kombiniert werden. Eine Transition ist *gültig*, falls

- der Ausgangszustand aktiv ist,
- das Ereignis `event` auftritt oder kein Ereignis angeben ist und
- die angegebene Bedingung `condition` wahr ist oder keine Bedingung gestellt wurde.

Sobald eine Transition gültig ist, wird die `conditionAction` ausgeführt. Die `transitionAction` wird beim Zustandsübergang ausgeführt. Für einfache Transitionen besteht zwischen den beiden Aktionen, abgesehen von der Reihenfolge, kein Unterschied.

Weitere Transitionstypen

Neben den erläuterten einfachen Transitionen gibt es in der Werkzeugleiste weitere Transitionstypen. Die *Standardtransition* (engl. *default transition*, s. Abb. 5.4) legt fest, welcher Zustand bei erstmaliger Ausführung eines Charts aktiv ist. Diese Festlegung ist zwingend erforderlich, weil grundsätzlich zu jedem Zeitpunkt der Ausführung genau ein Zustand aktiv sein muss (innerhalb einer Hierarchieebene). Dazu wird der entsprechende Knopf in der Werkzeugleiste gedrückt und dann der Rand des gewünschten Zustandes angeklickt.

Neben den genannten Transitionstypen gibt es noch die *innere Transition*, welche im Abschnitt 5.2 erläutert wird.

Hinweis: Es können auch mehrere Standardtransitionen innerhalb einer Hierarchieebene verwendet werden. Dabei ist aber mit Ereignissen oder Bedingungen sicherzustellen, dass jeweils genau eine der Standardtransitionen gültig ist.

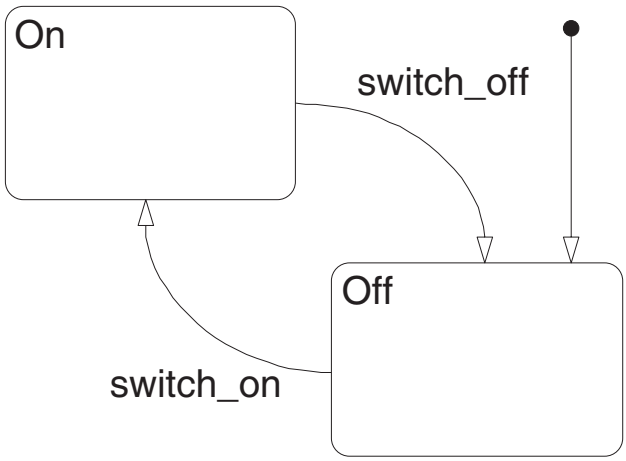


Abbildung 5.4: Chart mit Standardtransition

Verbindungspunkte

Ein weiteres Element ist der *Verbindungspunkt* (engl. *connective junction*). Mit Verbindungspunkten lassen sich komplexere Transitionen durch Zusammenführung und Verzweigung erzeugen. Damit lassen sich Verzweigungen, Fallunterscheidungen, zustandsfreie Flussdiagramme, verschiedene Schleifen, Selbstscheifen u.v.m. erzeugen (siehe Abb. 5.5).

Variablendeklaration

Alle verwendeten Variablen und Ereignisse müssen deklariert werden. Hierzu dient der *Stateflow-Explorer* (siehe Abb. 5.6) im Tools-Menü. Auf der linken Seite des Explorers befindet sich ein hierarchischer Baum, der alle Zustände *aller momentan offenen* State Charts enthält.

Die Deklaration erfolgt in der von objektorientierten Programmiersprachen bekannten Weise. Jedes Objekt kann seine eigenen Deklarationen besitzen. Diese sind dann jedoch nur im jeweiligen Mutterobjekt und den Kindobjekten *sichtbar* (es sei denn, in einem Kindobjekt ist eine Variable gleichen Namens definiert[1]). Auch die *Lebensdauer* der Variablen hängt vom Aktivierungszustand des Mutterobjektes ab. Ist ein Zustand nicht mehr aktiv, sind die in ihm deklarierten Variablen nicht mehr vorhanden („zerstört“).

Variablen und Ereignisse können als *lokal* oder als *Ein-/Ausgänge* von und nach SIMULINK® deklariert werden, wobei letztere nur in der obersten Ebene eines Charts vorkommen können.

Mit dem Menüpunkt „Add“ kann eine Variable (`data`) oder ein Ereignis (`event`)hinzugefügt werden. Bei Doppelklick auf das Symbol links neben dem Namen öffnet sich ein Dialog, bei dem weitere Optionen festgelegt werden können.

Bei *Ereignissen* lauten die wichtigsten Einstellmöglichkeiten:

Name: Name des Ereignisses

Scope: Ereignis ist lokal oder Ein-/Ausgang nach Simulink (nur in der Chart-Ebene).

[1] In diesem Fall wird die Variable des Kindobjektes angesprochen. Jedoch führt eine doppelte Vergabe von Variablenbezeichnungen meist zu Unübersichtlichkeiten und sollte generell vermieden werden.

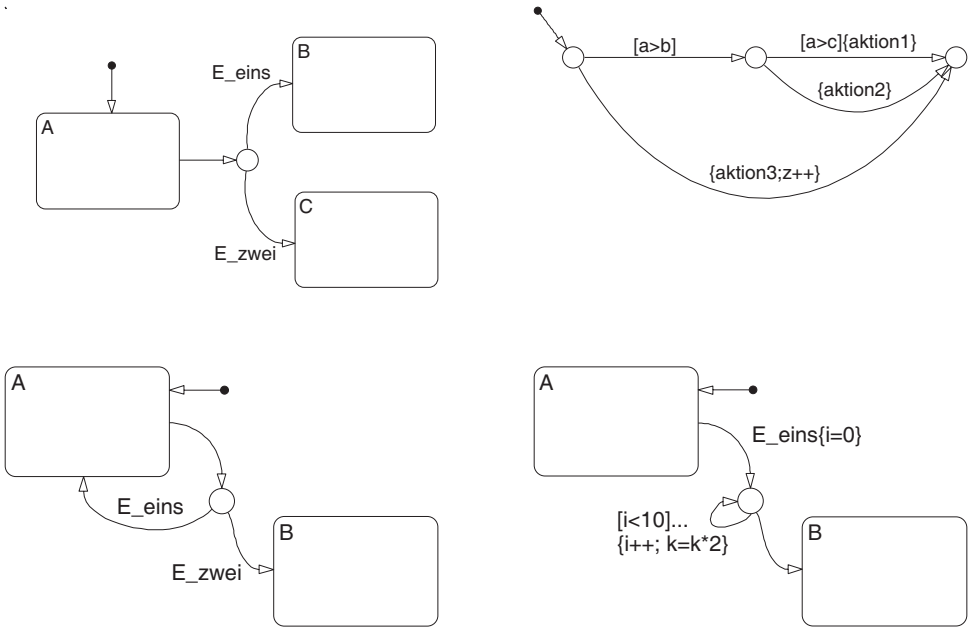


Abbildung 5.5: Komplexe Transitionen durch den Einsatz von Verbindungspunkten, links oben: Verzweigung, rechts oben: Flussdiagramm mit if-Abfrage, links unten: Selbstschleife, rechts unten: for-Schleife

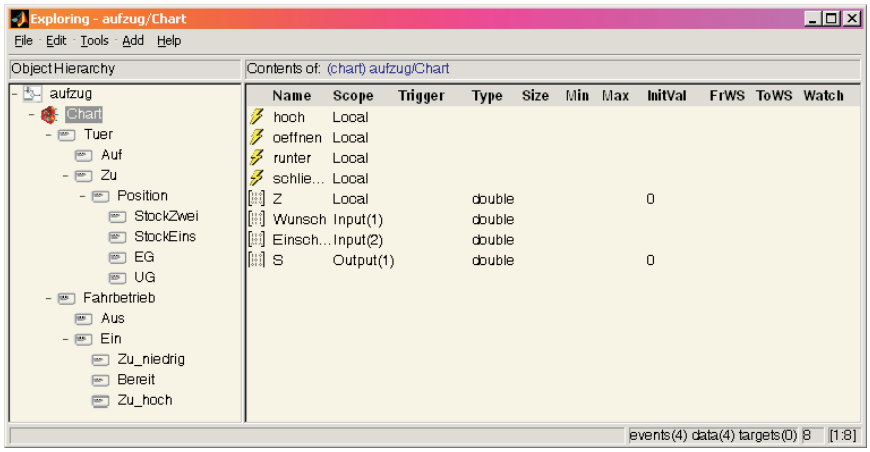


Abbildung 5.6: Der Stateflow-Explorer

Port: Nur bei Ausgang: Nummer des Port des Chart-Blockes in Simulink, an dem das Ereignis abgegriffen werden kann.

Index: Nur bei Eingang: Index des Vektorsignals am Trigger-Eingang des Chart-Blockes in Simulink, welches dieses Ereignis auslösen soll.

Trigger: Nur bei Ein-/Ausgang: Legt fest, ob eine steigende (von - nach +), eine fallende (von + nach -) oder beide Signalflanken ein Ereignis auslösen. (Wichtig: In jedem Fall muss die Null gekreuzt werden. Ein Übergang von 1 nach 2 löst kein Ereignis aus).
Die Signalflanken werden nur am Anfang eines Simulationsschritts ausgewertet. Mit der Option `Function Call` kann ein Ereignis auch innerhalb eines Simulationsschritts an einen Simulink-Block weitergegeben bzw. empfangen werden.

Bei *Variablen* lauten die wichtigsten Einstellmöglichkeiten:

Name: Name der Variablen

Scope: Variable ist lokal, konstant oder Ein-/Ausgang nach Simulink (nur in der Chart-Ebene). Eine Konstante kann nicht durch Stateflow-Aktionen verändert werden.

Port: Nur bei Ein-/Ausgang: Nummer des Port des Chart-Blockes in Simulink, an dem die Variable anliegt.

Type: Typ der Variablen (`double`,`single`,`int32`,...).

Initialize from: Die Variable kann durch einen hier eingegebenen Wert („data dictionary“ auswählen und Initialisierungswert ins Feld rechts daneben eintragen) oder aus dem Matlab-Workspace („workspace“ auswählen) zum Simulationsbeginn initialisiert werden.

Werden Ein-/Ausgänge festgelegt, tauchen im Simulink-Modell Ein- bzw. Ausgangsports beim Chart-Symbol auf. Zu beachten ist, dass alle *Eingangsereignisse* über den *Triggereingang* als Vektorsignal zugeführt werden müssen (vgl. Abb. 5.7).

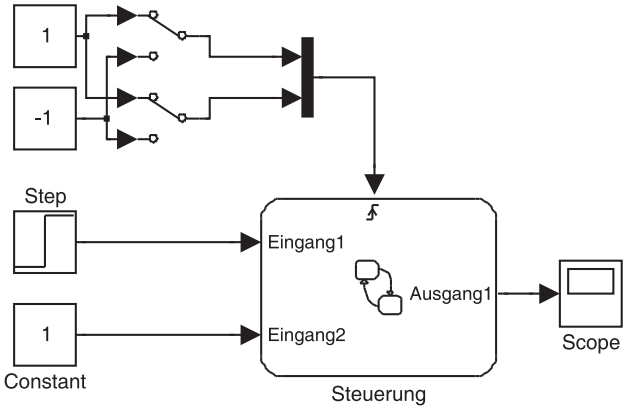


Abbildung 5.7: Chart mit verschiedenen Eingängen: Ereignisse werden als Vektor in den Triggereingang eingespeist, Variablen benutzen Eingangsports.

Neben Variablen und Ereignissen kann auch die *Aktivität eines Zustandes* als Ausgang nach Simulink verwendet werden. Einen solchen Ausgang legt man nicht im Explorer, sondern in der Eigenschaftsdialogbox des Zustandes an (Rechtsklick auf den Zustand, dann `Properties` im Kontextmenü). Nachdem man dort das Kästchen `Output State Activity` markiert, erscheint ein zusätzlicher Ausgangsport beim Chart-Symbol in Simulink und im Stateflow-Explorer. Der Wert dieses Ausgangs beträgt während der Simulation Eins, falls der Zustand aktiviert ist, sonst Null.

Hinweis: Ereignisse können durch die Aktion `/Ereignisname;` (bzw. {`Ereignisname;`} in Zuständen und Transitionen ausgelöst werden.

Hinweis: Die aktuelle Simulationszeit ist Inhalt der Variable „t“, welche *nicht* deklariert werden muss.

Charts ausführen

Stateflow-Charts sind in ein Simulink Modell eingebettet, welches während der Simulationszeit den Aufruf des Charts auslöst. In der Eigenschaftsdialogbox des Charts (Menü `File - Chart Properties`) ist neben weiteren Optionen festzulegen, unter welchen Bedingungen der Chart aufgerufen wird. Diese Eigenschaft wird unter dem Punkt `Update method` eingestellt (vgl. Abb. 5.8).

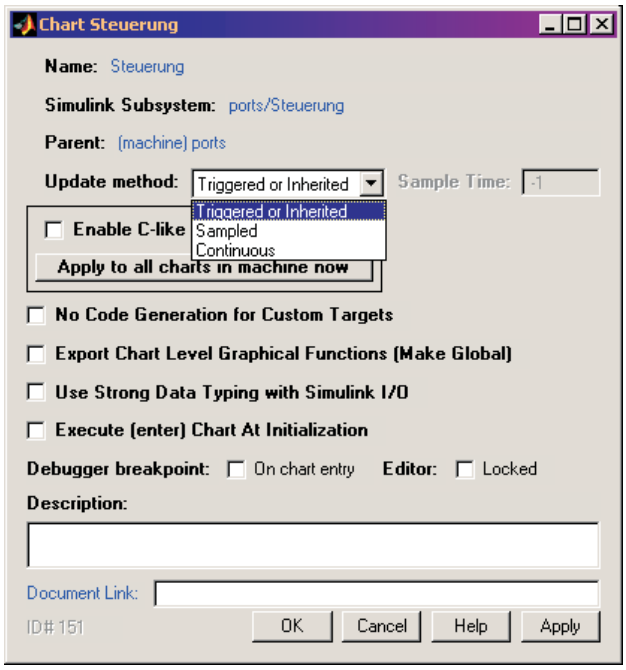


Abbildung 5.8: Einstellung der Update-Methode in den Chart Properties

Der Chart wird ausgeführt,

Triggered: wenn durch den Triggereingang des Chart-Blockes in Simulink ein Eingangsereignis ausgelöst wird,

Inherited: wenn durch die vorangehenden Blöcke eine Eingangsvariable neu berechnet wird (falls keine Eingangsereignisse deklariert sind),

Sampled: in festen Zeitabständen (unabhängig vom Rest des Simulinkmodells). Dieser Zeitabstand wird im Feld `Sample Time` festgelegt.

Continuous: bei jedem Integrationsschritt (auch bei Minor Steps).

Ein Einschalten der Option `Execute (enter) Chart At Initialization` bewirkt, dass beim Simulationsstart das Chart einmal ausgeführt wird, unabhängig von der gewählten Aktualisierungsmethode.

Die Simulation kann wie üblich in Simulink oder im Stateflow-Editor durch Drücken des Play-Buttons oder im Menü `Simulation` gestartet werden. Bei erstmaliger Simulation oder nach Veränderungen des Charts wird zuerst das Chart durch Kompilieren in eine s-function umgewandelt. Dieser Schritt kann relativ zeitaufwendig sein und zu einer merklichen Verzögerung führen, bis die Simulation abläuft.

Ist der Stateflow-Editor während der Simulationszeit offen, wird der Ablauf der Zustandsaktivierungen graphisch animiert. Zusätzliche Verzögerungen zur besseren Sichtbarkeit können im Debugger (Menü `Tools - Debugger`) konfiguriert werden. Hier kann die Animation auch gänzlich abgeschaltet werden. Viele weitere Funktionen unterstützen dort den Anwender bei der Fehlersuche. So können *Breakpoints* an verschiedenen Stellen gesetzt werden, automatisch Fehler (Zustandsinkonsistenzen, Konflikte, Mehrdeutigkeiten, Gültigkeitsbereiche, Zyklen) gesucht werden, zur Laufzeit Werte von Variablen und aktive Zustände angezeigt werden.

Hinweis: Die übergeordneten Simulationseigenschaften (Simulationsdauer, Integrationsschrittweiten, etc.) werden durch das Simulink-Modell festgelegt.

Aufgaben

5.1 ☞ Sie kennen nun alle Elemente, um einfache Zustandsautomaten zu simulieren. Experimentieren sie selbst: Erstellen Sie einfache Charts und lassen diese ablaufen. Wenn nötig, orientieren sie sich an den bisher abgebildeten Beispielen.

5.2 ☞ Erstellen Sie das Modell eines S/R-Flipflops. Dieses besitzt zwei Ereigniseingänge `Set` und `Reset` und einen binären Ausgang `Q`. Falls ein Set-Ereignis eintritt, wird der Ausgang auf 1 gesetzt (oder bleibt auf 1). Tritt ein Reset-Ereignis ein, wird der Ausgang auf 0 gesetzt (oder bleibt dort). Erproben Sie anschließend die Funktion in einem geeigneten Simulink-Modell.

5.3 ☞ Erstellen Sie ein diskretes Modell für einen Aufzug über vier Stockwerke (UG, EG, Stock1, Stock2). Sehen Sie für jedes Stockwerk einen entsprechend benannten Zustand vor. Vorerst soll die Steuerung des Aufzuges vereinfacht erfolgen: Erzeugen Sie mit Hilfe von Schaltern in Simulink die Ereignisse „Rauf“ und „Runter“, welche einen entsprechenden Zustandswechsel bewirken sollen. Die aktuelle Position des Aufzuges soll an Simulink übergeben werden.
Speichern Sie das Modell unter dem Namen `aufzug.mdl` und testen Sie es.

5.2 Weitere Strukturen und Funktionen

Neben den bisher genannten Basiselementen gibt es weitere Strukturen, welche komplexere Funktionen, aber auch die einfachere und übersichtlichere Darstellung von ereignisdiskreten Systemen ermöglichen.

Superstates und Subcharts

Mit *Superstates* lassen sich zusammengehörige Zustände zu einem Oberzustand zusammenfassen. Dies geschieht dadurch, dass zuerst mit dem State-Tool ein neuer Zustand im Stateflow-Editor angelegt wird. Dann wird durch Klicken und Ziehen an den Ecken des Zustandes (der Mauszeiger verwandelt sich in einen Doppelpfeil) der Zustand soweit vergrössert, bis er alle gewünschten Unterzustände umfasst (siehe Abb. 5.9).

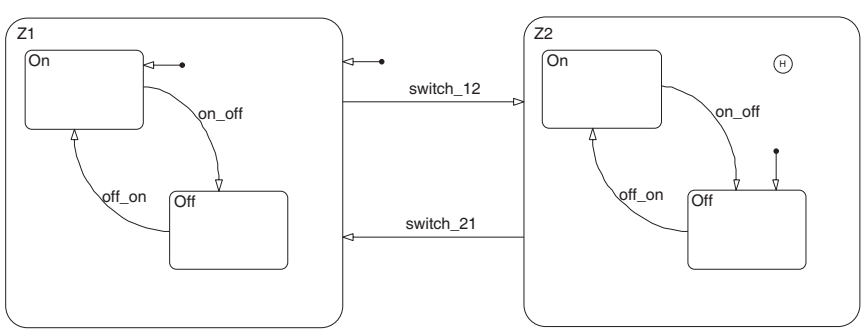


Abbildung 5.9: Chart mit zwei Superstates. Der rechte Superstate Z2 besitzt eine History Junction.

Für das Innere eines solchen Superstates gelten dieselben Regeln wie für die oberste Ebene (Initialisierung, Anzahl aktiver Zustände etc.). Ein Superstate kann wie ein herkömmlicher Zustand Ziel und Ausgangspunkt von Transitionen sein. Er wird dann aktiviert, wenn er selbst oder einer seiner Unterzustände das Ziel einer gültigen Transition ist. Ist ein Superstate nicht aktiviert, ist auch keiner seiner Unterzustände aktiv. Die Unterzustände können dabei als eigenständiges Chart innerhalb des Superstates aufgefasst werden.

Die Aufteilung in Superstates und Unterzustände kann jederzeit durch Änderung der Größe und Lage der Zustände verändert werden. Um eine unabsichtliche Veränderung zu verhindern, kann der Inhalt eines Zustandes *gruppiert* werden. Dazu doppelklickt man entweder in das Innere eines Zustandes oder wählt im Kontextmenü (Rechtsklick) des Zustandes den Punkt `Make Contents - Grouped` aus. Der Zustand wird dann grau eingefärbt.

Bei gruppierten Zuständen bleiben alle Unterzustände und Transitionen sichtbar, können aber nicht mehr bearbeitet werden. Wird im Kontextmenü der Punkt `Make Contents - Subcharted` ausgewählt, wird der Inhalt verdeckt. Damit lassen sich Charts mit komplizierten Teilfunktionen übersichtlicher darstellen. Die Funktion bleibt allerdings gegenüber gruppierten Superstates dieselbe. Durch Doppelklick kann ein Subchart geöffnet und dann bearbeitet werden. Zurück in die höhere Hierarchiebene kommt man mit dem „Nach-Oben-Pfeil“ in der Symbolleiste.

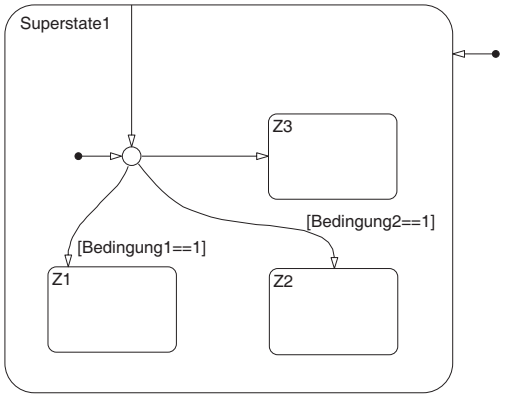


Abbildung 5.10: Superstate mit einer Inneren Transition

Verliert ein Superstate durch eine Transition seine Aktivität, werden auch alle Unterzustände passiv. Bei einer neuerlichen Aktivierung des Superstate entscheidet dann wieder die Standardtran-

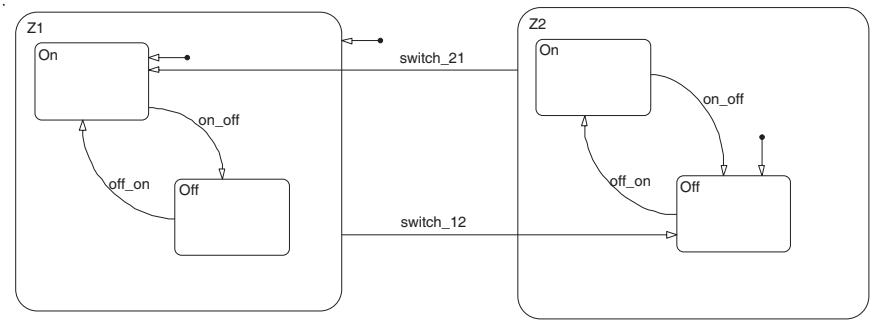


Abbildung 5.11: Zustandsdiagramm mit hierarchieübergreifenden Transitionen

sition, welcher Unterzustand die Aktivierung erhält. Dies kann man dadurch verhindern, indem man in den Zustand aus der Werkzeugleiste eine *History Junction* setzt. Nun erhält derjenige Unterzustand, welcher vor dem Verlust aktiviert war, die Aktivierung zurück (vgl. Abb. 5.9).

In einem Superstate kann eine weiterer Transitionstyp eingesetzt werden, die *Innere Transition*. Sie wird erzeugt, in dem eine Transition vom Rand des Superstates zu einem seiner Unterobjekte mit der Maus gezogen wird. Diese Transition wird immer auf Gültigkeit überprüft solange der Superstate aktiv ist, unabhängig von seinen Unterzuständen. Ein Beispiel für die Verwendung einer Inneren Transition ist die Vereinigung einer verzweigten Zustandskette. Anstatt Transitionen von allen Enden dieser Kette einzuzeichnen, genügt oft der Einsatz einer Inneren Transition (vgl. Abb. 5.10).

Hierarchieübergreifende Transitionen und Wurmlöcher

Transitionen können auch übergreifend über alle Hierarchien angelegt werden. So sind alle Verbindungen zwischen Superstates und Unterzuständen möglich. (siehe Abb. 5.11). Jedoch bleibt die Regel gültig, dass bei Zuständen in exklusiver Oder-Anordnung immer genau ein Zustand pro Superstate aktiv ist.

Hat man einen Zustand als Subchart anlegt, ist dessen Inhalt verdeckt. Mit Hilfe eines *Wurmloches* können auch Transitionen von und zu den Inhalten eines Subchartes angelegt werden. Zieht man eine Transition auf die Fläche eines Subchartes, wird in der Mitte ein Wurmloch angezeigt. Bewegt man sich mit der Maus bei gedrückter Maustaste auf dieses Wurmloch, wird der Subchart geöffnet

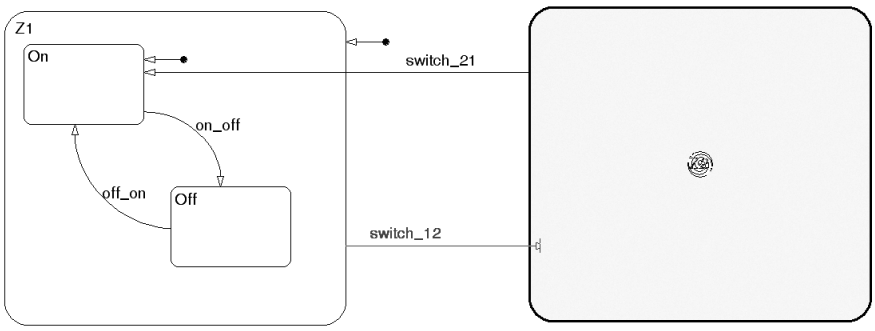


Abbildung 5.12: Zustandsdiagramm mit Transitionen, welche in ein Subchart hineinführen.

und man kann die Transition dort anbringen. Auch in umgekehrter Richtung ist der Vorgang möglich. Hierzu zieht man eine Transition vom Inneren eines Subcharts auf dessen Äußeres, so dass dort ein Wurmloch sichtbar wird. Ein Beispiel solcher Transitionen ist in Abb. 5.12 dargestellt.

Transitionsprioritäten

Je nach Beschriftung der Transitionen durch die Labels kann es zu Fällen kommen, bei denen zwei unterschiedliche Transitionen prinzipiell gleichzeitig gültig werden. Solche Mehrdeutigkeiten sollte man zwar bei der Charterstellung unbedingt vermeiden, jedoch gibt es für eine Auflösung dieses Konflikts feste Regeln.

So wird diejenige Transition ausgeführt, welche

- zu einem Zielzustand höherer Hierarchie führt,
- ein Label mit Ereignis- und Bedingungsabfrage besitzt,
- ein Label mit Ereignisabfrage besitzt,
- ein Label mit Bedingungsabfrage besitzt,

Ist mit diesen Kriterien noch keine eindeutige Entscheidung möglich, wird die Reihenfolge graphisch bestimmt: Es wird diejenige Transition zuerst ausgeführt, deren Abgangspunkt der linken oberen Ecke eines Zustandes (bzw. der 12-Uhr-Stelle eines Verbindungspunktes) im Uhrzeigersinn als nächster folgt.

Abbildung 5.13: Chart mit parallelen Zuständen

Parallele Zustände

Bisher bezogen sich alle Ausführungen auf die exklusive Anordnung (Oder-Anordnung) von Zuständen, bei der immer genau ein Zustand aktiviert ist. Neben dieser Anordnung gibt es die *parallele Und-Anordnung*, bei der alle Zustände einer Hierarchieebene gleichzeitig aktiv sind und nacheinander abgearbeitet werden.

Innerhalb jeder Hierarchieebene (Chart, Superstate) kann die Art der Anordnung getrennt festgelegt werden. Dies geschieht im Kontextmenü eines zugehörigen Zustandes unter `Decomposition`. Wird auf `Parallel (AND)` umgeschaltet, ändert sich die Umrandung der betroffenen Zustände zu einer gestrichelten Linie und es wird eine Zahl in der rechten oberen Ecke eingeblendet (siehe Abb. 5.13). Diese Zahl gibt die Ausführungsreihenfolge der Zustände an. Die parallelen Zustände werden nacheinander von oben nach unten und von links nach rechts ausgeführt.

Mit parallelen Zuständen können Systeme modelliert werden, welche parallel ablaufende Teilprozesse besitzen.

Boxen

Ein weiteres Gruppierungsmittel stellen *Boxen* dar. Eine Box kann mit dem entsprechenden Werkzeug der Werkzeugleiste erzeugt werden. Auch kann ein Zustand mit seinem Kontextmenü mit dem Punkt `Type` in eine Box umgewandelt werden (und umgekehrt).

Mit einer Box können Bereiche eines Charts zur besseren Übersichtlichkeit eingerahmt werden (siehe Abb. 5.14). Zu beachten ist, dass die Box die Ausführungsreihenfolge verändern kann, sonst jedoch keinen Einfluss auf die Funktion eines Charts besitzt.

Auch Boxen können in der bekannten Weise durch den Eintrag `Make Contents` des Kontextmenüs gruppiert und zu einem Subchart umgewandelt werden, so dass Inhalte nicht mehr veränderbar bzw. sichtbar sind.

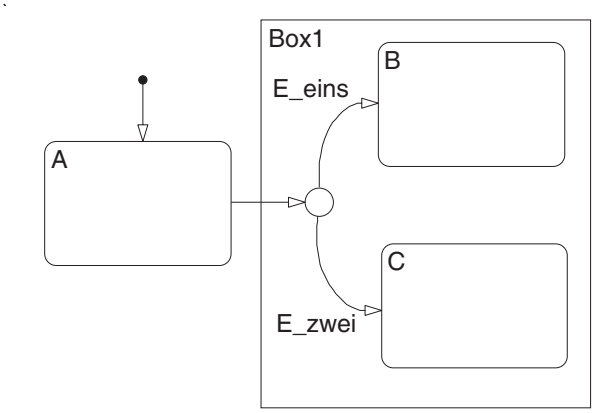


Abbildung 5.14: Zustandsdiagramm mit Box

Graphische Funktionen und Wahrheitstabellen

In der Werkzeugleiste gibt es Werkzeuge zur Erstellung von *graphischen Funktionen* und *Wahrheitstabellen*. Damit können skalare Funktionen mit skalaren Argumenten und Wahrheitstabellen definiert werden. Sollten Sie diese Elemente in Ihrem Modell benötigen, entnehmen Sie bitte die nötigen Informationen der Online-Hilfe oder der weiterführenden Literatur (siehe Abschnitt 5.4).

Aufgaben

5.4 ☞ Ergänzen Sie das Modell des Aufzuges um folgende Funktionen:

- Integrieren Sie einen Hauptschalter, mit dem der Aufzug ein- bzw. ausgeschaltet wird.
- Erweitern Sie die Steuerung nun derart, dass von Simulink ein Ziel-Stockwerk vorgegeben wird.
- Berücksichtigen Sie den Zustand der Türe (offen - geschlossen). Nach Ankunft des Aufzuges öffnet die Türe für eine gewisse Zeitspanne und schliesst danach wieder. Stellen Sie sicher, dass sich der Aufzug nur bei geschlossenen Türen bewegt.

Tipps: Verwenden Sie auf der obersten Ebene zwei parallele Zustände. Steuern Sie nun die „Rauf-“ und „Runter“ Ereignisse geeignet intern. Betten Sie die bisherigen Stockwerks-Zustände in einen Zustand `Tür zu` ein.

5.3 Action Language

In diesem Abschnitt werden die verschiedenen Elemente der Action Language kurz erläutert, welche in den Labels der Zustände und Transitionen benötigt werden.

Besondere Bedeutung kommt den in Tab. 5.1 genannten Schlüsselwörtern zu, welche nicht als Variablen- oder Ereignisnamen verwendet werden können.

Schlüsselwort **Abkürzung**	**Bedeutung**
`change(data_name)` `chg(data_name)`	Erzeugt ein lokales Event, wenn sich der Wert von `data_name` ändert.
`during` `du`	Darauf folgende Aktionen werden als *During Action* eines Zustandes ausgeführt.
`entry` `en`	Darauf folgende Aktionen werden als *Entry Action* eines Zustandes ausgeführt.
`entry(state_name)` `en(state_name)`	Erzeugt ein lokales Event, wenn der angegebene Zustand aktiviert wird.
`exit` `ex`	Darauf folgende Aktionen werden als *Exit Action* eines Zustandes ausgeführt.
`exit(state_name)` `ex(state_name)`	Erzeugt ein lokales Event, wenn der angegebene Zustand verlassen wird.
`in(state_name)`	Bedingung, welche als `true` ausgewertet wird, wenn der angegebene Zustand aktiv ist.
`on event_name`	Darauf folgende Aktionen werden ausgeführt, wenn das angegebene Ereignis auftritt.
`send(event_name,state_name)`	Sendet das angegebene Ereignis an den Zustand `state_name`.
`matlab(funktion,arg1,arg2,...)` `ml(funktion,arg1,arg2,...)`	Aufruf der angegebenen Matlab-Funktion mit den Argumenten `arg1`, `arg2`,...
`matlab.var` `ml.var`	Zugriff auf die Variable `var` des Matlab-Workspaces.

Tabelle 5.1: Schlüsselwörter der Action Language

In den Tabellen 5.2 bis 5.4 sind die in der Action Language definierten Operatoren zusammengestellt. Alle Operationen gelten dabei nur für *skalare* Größen. Ausnahmen bilden einige wenige Matrizenoperatoren, welche in Tabelle 5.5 zusammengestellt sind. Zu beachten ist, dass der Zugriff auf Elemente von Matrizen in anderer Weise erfolgt als in Matlab!

Wird auf eine Variable oder ein Ereignis zugegriffen, versucht Stateflow diese in derselben Hier-

archieebene zu finden, in der die aufrufende Aktion steht. Ist die Suche nicht erfolgreich, wird schrittweise in den darüberliegenden Hierarchieebenen weitergesucht. Soll auf ein Objekt eines anderen Mutter-Objekts zugegriffen werden, so muss der volle Pfad dorthin angegeben werden (Beispiel: `Superstate2.Zustand5.Anzahl`).

Operator	Beschreibung
`a + b`	Addition
`a - b`	Subtraktion
`a * b`	Multiplikation
`a / b`	Division
`a %% b`	Restwert Division (Modulus)

Tabelle 5.2: Numerische Operatoren

Operator	Beschreibung
`a == b`	Gleichheit
`a ~= b`	Ungleichheit
`a != b`	
`a > b`	Größer-Vergleich
`a < b`	Kleiner-Vergleich
`a >= b`	Größer-Gleich-Vergleich
`a <= b`	Kleiner-Gleich-Vergleich
`a && b`	Logisches UND
`a & b`	Bitweises UND
`a || b`	Logisches ODER
`a | b`	Bitweises ODER
`a ^ b`	Bitweises XOR

Tabelle 5.3: Logische Operatoren

Mit den temporalen Logikoperatoren in Tabelle 5.6 lassen sich Bedingung an ein mehrfaches Auftreten von Ereignissen knüpfen. Sie dürfen nur in Transitionen mit einem Zustand als Quelle und in Zustandsaktionen verwendet werden. Das Aufsummieren von Ereignissen erfolgt nur, solange der Quellzustand aktiv ist. Bei einem Zustandswechsel wird der Zähler auf Null zurückgesetzt.

5.4 Weiterführende Informationen

Die vorliegende Anleitung stellt nur einen kurzen Überblick über die wichtigsten Funktionen von STATEFLOW® dar, um ereignisdiskrete Systeme in SIMULINK®-Modellen einzubinden.

Sollten Sie weitere Informationen benötigen, ziehen Sie die Online-Hilfe und die Dokumentation von STATEFLOW® zu Rate. Weiterhin ist folgendes Buch, welches auch dieser Anleitung als Informationsquelle diente, empfehlenswert:

Operator	Beschreibung
`~a`	Bitweise Invertierung
`!a`	Logische NOT Operation
`-a`	Multiplikation mit -1
`a++`	Variable um 1 erhöhen
`a--`	Variable um 1 erniedrigen
`a = expression`	Zuweisung an die Variable
`a += expression`	identisch mit `a = a + expression`
`a -= expression`	identisch mit `a = a - expression`
`a *= expression`	identisch mit `a = a * expression`
`a /= expression`	identisch mit `a = a / expression`

Tabelle 5.4: Unäre Operatoren und Zuweisungen

Operator	Beschreibung
`matrix1 + matrix2`	Elementweise Addition zweier gleichgroßer Matrizen
`matrix = n`	Belegung aller Matrixelemente mit dem Skalar `n`
`matrix * n`	Multiplikation aller Matrixelemente mit dem Skalar `n`
`matrix[i][j]`	Zugriff auf das Element (`i`,`j`) der Matrix

Tabelle 5.5: Matrizenoperatoren

Operator	Beschreibung
`after(n,event)`	Wahr, wenn das Ereignis mindestens `n`-mal aufgetreten ist.
`before(n,event)`	Wahr, wenn das Ereignis weniger als `n`-mal aufgetreten ist.
`at(n,event)`	Wahr, wenn das Ereignis genau `n`-mal aufgetreten ist.
`every(n,event)`	Wahr, wenn das Ereignis genau `n`-mal, 2`n`-mal,... aufgetreten ist.

Tabelle 5.6: Temporale Logikoperatoren

A. Angermann, M. Beuschel, M. Rau, U. Wohlfarth:
MATLAB - SIMULINK - STATEFLOW
Grundlagen, Toolboxen, Beispiele
Oldenbourg Verlag, München
ISBN 3-486-27377-9
http://www.matlabbuch.de

6 Übungsaufgaben

6.1 ☞Ein System ist durch folgendes Differentialgleichungssystem beschrieben:

$$\dot{v}\,(t) = a_{11} \cdot v\,(t) + a_{12} \cdot p_A\,(t) + a_{13} \cdot p_B\,(t)$$

$$\dot{p}_A\,(t) = a_{21} \cdot v\,(t) + a_{22} \cdot p_A\,(t) + b_{12} \cdot u\,(t)$$

$$\dot{p}_B\,(t) = a_{31} \cdot v\,(t) + a_{33} \cdot p_B\,(t) + b_{13} \cdot u\,(t)$$

mit

$$\begin{array}{lll} a_{11} = -30 & a_{12} = 5 & a_{13} = -8 \\ a_{21} = -6 & a_{22} = -8 & b_{12} = 6 \\ a_{31} = 28 & a_{33} = -19 & b_{13} = -10 \end{array}$$

wobei $u\,(t)$ die Eingangs- und $v(t)$ die Ausgangsgröße ist.

1. Bestimmen Sie das Gleichungssystem zur Berechnung der stationären Arbeitspunkte des Systems, wenn u_A, v_A auf bekannt sind.
2. **Während der Übung:** Berechnen Sie die Arbeitspunkte für $u_A = 0, \quad u_A = 1, \quad v_A = 5.$
3. Führen Sie für dieses System Zustandsgrößen ein und geben Sie eine Zustandsdarstellung.
4. **Während der Übung:** Stellen Sie den zeitlichen Verlauf der Zustandsgrößen dar, falls an den Eingang $u(t)$ ein Einheitssprung gelegt wird.
5. Zeichnen Sie den Wirkungsplan des Systems. Benutzen Sie dafür nur proportional und integrierende Glieder.
6. **Während der Übung:** Simulieren Sie dieses System mit Simulink anhand des Wirkunsplans. Geben Sie dabei wieder einen Einheitsprung auf den Systemeingang $u(t)$.
7. **Während der Übung:** Schreiben Sie eine S-Funktion für das System und simulieren Sie eine Einheitssprungantwort mit Simulink.

6.2 ☞Adaptation einer Klausuraufgabe F98

Eine Stahlkugel soll in folgender Anordnung durch einen Elektromagneten im Schwebezustand gehalten werden.

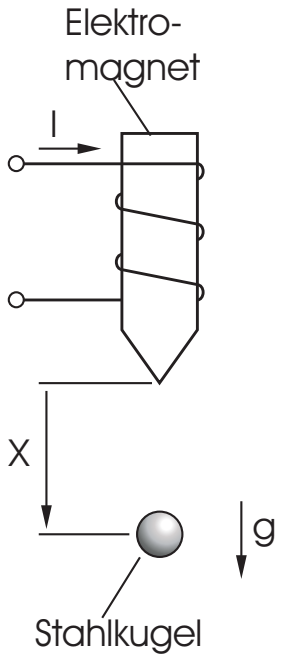


Abbildung 6.1: Skizze der Anordnung

Die vom Elektromagnet auf die Kugel ausgeübte Kraft F_m ist in erster Näherung durch folgende Gleichung gegeben:

$$F_m = a_1 \cdot \frac{I^2}{(a_0 + x)^2}$$

mit $a_0 = 6.9\ mm$ und $a_1 = 1.16 \cdot 10^{-4}\ N\,m^2/A^2$.

Die Masse der Kugel beträgt $m = 0.114\ kg$ und die Erdbeschleunigung $g = 9.81\ m/s^2$.

1. Stellen Sie die Differentialgleichung auf, die die Bewegung der Kugel im Magnetfeld beschreibt.
2. Welcher Strom I_A muss eingestellt werden, um die Kugel in einem Arbeitspunkt $X_A = 0.02$m zu halten?
3. Führen Sie für diesen Arbeitspunkt (X_A, I_A) eine Linearisierung der DGL durch und bringen Sie das System in Zustandsraumdarstellung. Geben Sie zusätzlich den Frequenzgang des Systemes an.
4. Ist das System stabil? Begründen Sie ihre Antwort anhand der Physik, der DGL und der linearen Systemdarstellungen. Bestätigen Sie ihre Antwort, indem Sie das nichtlineare System mit Simulink simulieren.
5. Finden Sie einen linearen Regler (P, PD, PID, PI), so dass das System stabilisiert wird. Überprüfen Sie ihre Wahl anhand der Simulation.

6.3 ☞ Vorbereitung zum Versuch „Schaltender Regler“

Für die Regelung einer Vielzahl von Systemen wird das Stellglied diskontinuierlich angesteuert. Der Grund dafür liegt darin, diskrete Stellglieder erheblich einfacher aufgebaut sind, und deshalb preislich sehr viel günstiger, wartungsärmer und fehlerunanfälliger sind.

Die experimentell ermittelte Übergangsfunktion einer Regelstrecke (System + Stellglied) wird näherungsweise durch die Übergangsfunktion eines PT_1T_T mit Totzeit angenährt. Die Zeitkonstante sei T_g, die Totzeit sei T_u (sogenantes T_uT_g-Modell) und der Übertragungsbeiwert sei K_s:

$$G(s) = \frac{K_s}{T_g \cdot s + 1} \cdot e^{-T_u \cdot s}$$

An der Regelstrecke wurden durch Messung folgende Streckenkennwerte ermittelt: $T_u = 22\ s$, $K_s = 55$ und $T_g = 111\ s$.

Dieses System wird mit einen Zweipunktregler geregelt wobei die Stellgrösse $u(t)$ nur die Werte 0 und 1 annehmen kann.

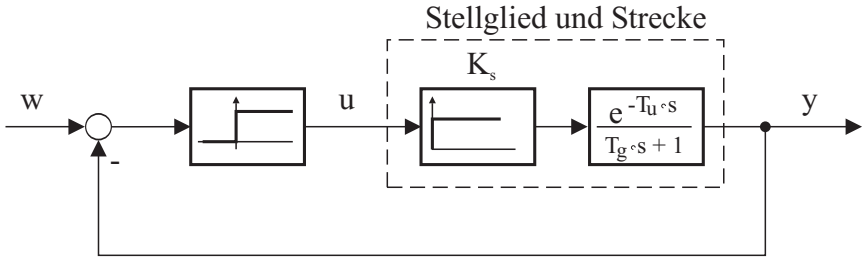


Abbildung 6.2: Zweipunktregler

1. Berechnen Sie die Antwort dieses Systems, wenn für den Eingang $w(t) = 1(t)$ gilt. Welche Periode und Amplitude ergeben sich dabei? Das System sei für $t \leq 0$ im Ruhezustand.
2. Was würde passieren, falls $T_u = 0$ wäre. Was würden Sie In diesem Fall vorschlagen, um das Relais-Element zu schonen?
3. Was vor allem ist bei dieser Regelungslösung unbefriedigend?

6.4 ☞

- Erstellen Sie ein Modell (vgl. Abb. 6.3) der nichtlinearen van-der-Pol[1] Differentialgleichung

$$\frac{d^2x}{dt^2} + \mu \cdot \left(x^2 - 1\right) \cdot \frac{dx}{dt} + \omega^2 \cdot x = 0$$

  wobei $\mu = 1$, $\omega = 1$, $x(0) = 1$ und $\dot{x} = 0$.

  Speichern Sie es unter dem Namen **vanpol**. Verwenden Sie die Blöcke Sum, Product, Fcn, Gain, Integrator, Mux, Scope und Outport. Achten Sie beim Erstellen auch darauf, bei welchen Blöcken der Name und wenn ja, welcher, angezeigt wird. Dieses Modell ist wie viele andere bei den **Examples und Demos** von MATLAB/SIMULINK hinterlegt.

[1] Balthazar van der Pol (1889-1959): Hollandischer Elektroingenieur

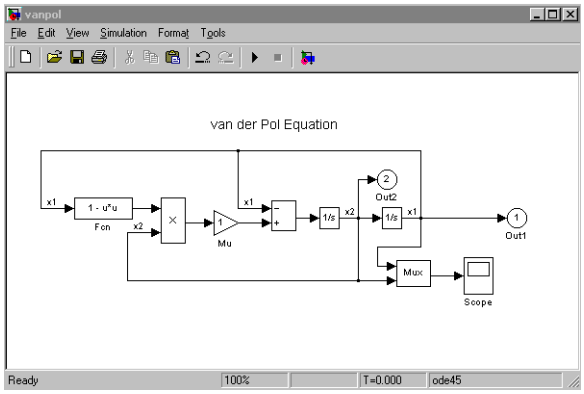


Abbildung 6.3: Modell der van-der-Pol Differentialgleichung

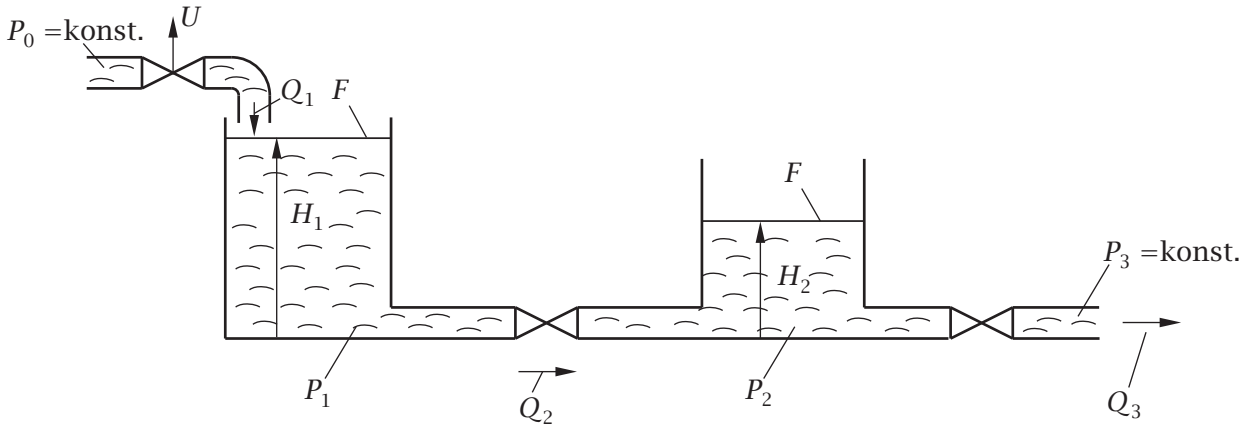
- Benutzen Sie die Hilfe-Funktion um x1 im Integrierer zwischen -1 und 1 zu begrenzen. Simulieren Sie das Ergebnis.
- Schalten Sie die Begrenzung wieder aus. Benutzen Sie erneut die Hilfe-Funktion, um x1 im Integrierer zurückzusetzen (reset), sobald $x_2 = \dot{x}$ einen Nulldurchgang mit positiver Steigung besitzt. Simulieren Sie das Ergebnis.
- Ändern Sie die Formate wie Farben, Schriftgrößen, ...

6.5 ☞ Zustandsregelung einer Druckregelstrecke (vgl. MRT 10.1T)

Zwei Flüssigkeitsbehälter mit gleichen Querschnitten F sind über eine Rohrleitung mit Drosselstelle verbunden. Für die Drosselstellen gelten die **nichtlinearen** Durchflussgleichungen

$$Q_1 = c_1 \cdot U \ , \quad Q_2 = c_2 \cdot \sqrt{P_1 - P_2} \ , \quad Q_3 = c_3 \cdot \sqrt{P_2} \ .$$

Die Dichte der Flüssigkeit ist ρ. Verwenden Sie folgende Parameterwerte:
$F = 12\,\mathrm{cm}^2,\ \rho = 10^{-3}\,\mathrm{kg/cm}^3$
$c_1 = 15,0\,\mathrm{cm}^3/\mathrm{s},\ c_2 = 5,0\,\mathrm{cm}^3/\mathrm{s},\ c_3 = 4,0\,\mathrm{cm}^3/\mathrm{s}$



(a) Erstellen Sie ein nichtlineares Simulink-Modell mit der Ausgangsgröße $Y = P_2$ und der Eingangsgröße U. Starten Sie eine Simulation der Dauer $t_{sim} = 300\,\mathrm{s}$, während am Eingang U eine Einheitssprung anliegt, und betrachten Sie Ein-, Ausgang und die Zustände in einem Scope.

(b) Bestimmen Sie die Zustände $[H_{1A},\ H_{2A}]^T$ und den Eingang U_A in einem Arbeitspunkt mit $Y_A = 15\,\mathrm{hPa}$.

(c) Linearisieren Sie das Modell um diesen Arbeitspunkt und bringen Sie das System in Zustandsraumdarstellung.

(d) Bestimmen Sie die Eigenwerte der Regelstrecke.

(e) Bestimmen Sie den Rückführvektor

$$\mathbf{k} = \begin{bmatrix} k_1 \\ k_2 \end{bmatrix}$$

einer Zustandsrückführung so, dass die Zeitkonstanten um den Faktor 6 reduziert werden.

(f) Bestimmen Sie eine Vorsteuermatrix S so, dass im Arbeitspunkt stationäre Genauigkeit erreicht wird.

(g) Implementieren Sie die Zustandsrückführung und die Vorsteuerung im Simulink-Modell und überprüfen Sie Ihre Ergebnisse.