Zusammenfassung von MATLAB-Anweisungen

M. Schubert

Hochschule Regensburg

1 Einführung

Rechnergestützter Schaltungsentwurf auf Systemebene geschieht zunehmend mit komplexen Werkzeugen, wobei MATLAB einen bedeutenden Marktanteil erringen konnte.

MATLAB ist ein Akronym für MATrix LABoratory und bezeichnet ein Softwarepaket für numerische Mathematik.

Diese Zusammenfassung hält sich eng an Referenz [1], entstanden am Lehrstuhl für Elektrische Antriebssysteme der Technischen Universität München. Weitere Infos findet man unter www.matlabbuch.de.

Zeichen und Symbole für diese Kurzfassung:

cll → ... Doppelklick mit der linken Maustaste auf ...

2 Matlab Grundlagen

2.1 Erste Schritte mit Matlab

Starten von Matlab mit Doppelclick der linken Maustaste (**`cll`**) auf das Matlab Symbol oder die Datei **`matlab.exe`**.

Beenden von Matlab durch Eingabe von **`quit`**, **`exit`** oder **`Strg+Q`**.

2.1.1 Matlab Desktop

Command-Window (immer vorhanden)

Eingabe der Kommandos nach dem Matlab-Prompt: **`>>`**. Nach einem **`%`**-Zeichen ist der Rest der Zeile Kommentar.

Default-Variable ist **`ans`** (Answer)

```
>> (40^3 + 3*2e3)/7          % Eingabe des Anwenders
ans =                        % Reaktion von Matlab
       1000                  % Reaktion von Matlab
```

Editor (immer vorhanden)

Öffnen durch Eingabe von **`edit`** im Command-Window, ermöglich die Erstellung von M-Dateien. Jede Eingabe im Command-Window kann auch in eine M-Datei geschrieben und dann durch Eingabe des Dateinamens als Skript abgearbeitet werden.

Command History (nur bei neueren Matlab-Versionen)

Listet die im Command-Window eingegebenen Befehle auf. Die Befehle lassen sich durch ↑ und ↓ scrollen, durch **`cll`** wiederholen oder mit Copy&Paste in eine M-Datei kopieren und noch einmal abarbeiten.

Workspace Browser (nur bei neueren Matlab-Versionen)

Zeigt den Inhalt des aktuellen Workspace an, also definiert Variablen und deren Struktur, etc.

Current Directory Browser (nur bei neueren Matlab-Versionen)

Zeigt das aktuelle Verzeichnis an (z.B. ein Listing der dort befindlichen M-Dateien).

Profiler (nur bei neueren Matlab-Versionen)

Zeigt die Verwendung und den CPU-Zeitverbrauch einzelner Befehle in Matlab-Skripts.

Shortcut-Leiste (versionsabhängig)

Ermöglich den Aufruf häufig benötigter Kommandos durch Mausklick (**`cll`**).

2.1.2 Matlab Hilfe

Tabelle 2.1.2: Hilfe in Matlab

Zeichen	Beschreibung
`help [Befehl]`	Online-Hilfe im Command Window [zu einem bestimmten Befehl]
`helpwin [Befehl]`	Matlab Online-Hilfe im Hilfe-Browser [zu einem bestimmten Befehl]
`doc [Befehl]`	HTML-Dokumentation im Hilfe-Browser [zu einem bestimmten Befehl]
`lookfor suchstring`	Suche nach suchstring in der ersten zeile aller Matlab-Dateien im Matlab-Pfad

2.1.3 Zuweisungen

Die Deklaration von Variablen geschieht mit Hilfe des Zuweisungszeichen "=".

```
>> A = 5*3
A =
    15
```

Eine existierende Variable ist im Workspace gespeichert und sichtbar und kann durch Eingabe des namens im Command-Window abgefragt werden:

```
>> A
A =
    15
```

Ein Semikolon am Zeilenende unterdrückt das Echo des Kommandos:

```
>> a = 2
```

Matlab ist case-sensitiv, also a≠A. Erlaubt sind bis zu 63 Buchstaben, das "_" und Zahlen, diese aber nicht am Namensanfang.

```
>> B = a * A
B =
     30
```

Werden Matlab-Anweisungen in einer Zeile durch "," getrennt, dann hat jede Anweisung, die mit einem Komma abgeschlossen wurde, ein Echo (wie bei Return):

```
>> x = 2; y = 3; z=x*y;  % kein Echo
>> x = 2, y = 3; z=x*y   % kein Echo für y = 3;
x =
    2
z = 6
```

Namen, die es im Workspace nicht gibt, können nicht abgefragt werden.

```
>> u                                          % u git es nicht
??? Undefined function or variable 'u'.
```

Einige Variablennamen sind reserviert, z.B. **ans**, **eps**, **i**, **j**, **pi**, **NaN**. Reservierte Namen können eingegeben werden und erhalten eine Antwort, z.B. die Fließkommagenauigkeit **eps**. Die Buchstaben **i**, **j** stehen für $\sqrt{-1}$, das Kürzel **NaN** steht für "Not a Number" und **inf** für "infinite", was man z.B. als Reaktion auf die Eingabe **1/0** erhält.

```
>> eps, pi, NaN
ans =
      2.2204e-016
ans =
      3.1416
ans =
      NaN
>> 1/0
Warning: Divide by zero
ans =
      inf
```

Die Buchstaben **i** und **j** zeigen den Imaginärteil einer Zahl an.

```
>> complex_number = sqrt(-4)     % Wurzel aus -4
complex_number =
        0 + 2.0000i
```

Anmerkungen:

1. Für die Euler'sche Zahl e ist keine Variable reserviert, man erhält sie mit **exp(1)**.
2. Variablennamen und Funktionsnamen müssen sich unterscheiden. Im Zweifelsfalle Namen mit **which -all <*name*>** abfragen

2.1.4 Mathematische Funktionen und Operatoren

Mathematische Funktionen und Operatoren			
+ - * / ^	Rechenoperatoren	exp(x)	Exponentialfunktion e^x
mod(*x***,** *x***)**	x Modulo y	log(x)	natürl. Logarithmus
rem(*x***,** *x***)**	Rest nach Division x/y	log10(x)	Zehner-Logarithmus
sqrt(*x***)**	Quadratwurzel	erf(x)	Error-Funktion
abs(*x***)**	Betrag	real(x)	Realteil
sign(*x***)**	Signaum (Vorzeichen, {-1, 0, 0})	imag(x)	Imaginärteil
round(*x***)**	Runden	conj(x)	kompl. Kunjugation
ceil(*x***)**	Aufrunden	angle(x)	Phase kompl. Zahl
floor(*x***)**	Abrunden		
Trigonometrische Funktionen			
sin(*x***)**	Sinus	atan(x)	Arcus-Tangens ± π/2
cos(*x***)**	Cosinus	atan2(x)	Arcus-Tangens ± π
tan(*x***)**	Tangens	sinh(x)	Sinus Hyperbolicus
cot(*x***)**	Cotangens	sinc(x)	sin(πx)/(πx)

Tabelle 2.1.4: Mathematische Funktionen und Operatoren in Matlab

2.2 Variablen

2.2.1 Datentypen in Matlab

Anweisung	**Beschreibung**
double(*x***)**	Umwandlung Real-Typ, double precision (64 Bit). **Default-Typ.**
single(*x***)**	Umwandlung Real-Typ, single precision (32 Bit)
int8(*x***)**	Umwandlung in Integer-Zahl mit 8 Bit
int16(*x***)**	Umwandlung in Integer-Zahl mit 16 Bit
int32(*x***)**	Umwandlung in Integer-Zahl mit 32 Bit
uint8(*x***)**	Umwandlung in positive 8 Bit Integer-Zahl (ohne Vorzeichenbit)
uint16(*x***)**	Umwandlung in positive 16 Bit Integer-Zahl (ohne Vorzeichenbit)
uint32(*x***)**	Umwandlung in positive 32 Bit Integer-Zahl (ohne Vorzeichenbit)
char(*x***)**	Umwandlung in ein Zeichen
logical(*x***)**	Umwandlung in einen logischen Wert

Tabelle 2.2.1: Datentypen und entsprechende Umrechnungsoperationen in Matlab. Ohne Angabe eines Datentyps sind Zahlenwerte vom Typ **double**.

2.2.2 Vektoren und Matrizen

Anweisung	**Beschreibung**
[*x11 x12 ... ; x21 x22 ...***]**	Eingabe von Vektoren und Matrizen
Startwert **:** *[Schrittweite***:***] Zielw.*	"**:**"-Operator erzeugt Zeilenvektor, Default-Schritt ist 1
linspace(*Start***,** *Ziel***,** *Anzahl***)**	Erzeugung linearer Vektoren mit *Anzahl* Elementen
logspace(*Start***,** *Ziel***,** *Anzahl***)**	Erzeugung logarithm. Vektoren mit *Anzahl* Elementen
eye(*n***)**	Einheitsmatrix der Größe n x n, $a_{i,j}$=1 für i=j, sonst 0
zeros(*Zeilen***,** *Spalten [***,***Typ]***)**	*Zeilen* x *Spalten* Matrix, alle Elemente $a_{i,j}$=0
ones(*Zeilen***,** *Spalten [***,***Typ]***)**	*Zeilen* x *Spalten* Matrix, alle Elemente $a_{i,j}$=1
rand(*Zeilen***,** *Spalten***)**	*Zeilen* x *Spalten* Matrix, $0 \le a_{i,j} \le 1$ gleichvert. Zufallswerte
randn(*Zeilen***,** *Spalten***)**	*Zeilen* x *Spalten* Matrix, normalvert. Zufallswerte (Gauß)

Tabelle 2.2.2: Sprachelemente zur Erzeugung von Vektoren und Matrizen.

Die einfachste Eingabe von Vektoren und Matrizen ist die Verwendung eckiger Klammern [ ], wobei Elemente einer Zeile durch Kommas oder Freizeichen getrennt werden. Eine neue Zeile beginnt nach einem Semikolon.

Beispiel für die Verwendung von Leerzeichen, Kommas und Semikolon als Trennzeichen:

```
>> mein_vector = [1 2 3]   % Zeilenelemente ohne Kommas
>> mein_vector = [1, 2, 3] % Elemente mit Kommas, gleichwertig
mein_vector =
     1     2     3

>> mein_vector = [1; 2; 3]   % 3 Zeilen, da Semicolons trennen
mein_vector =
     1
     2
     3
```

Vektoren mit fortlaufenden Elementen können mit dem "**:**"-Operator erzeugt werden:

```
Startwert [: Schrittweite] : Zielwert
```

Der Default-Wert bei fehlender Schrittweite ist +1. Beispiele:

```
>> OMEG = 0:0.5:pi
OMEG =
    0   0.5000   1.0000   1.5000   2.0000   2.5000   3.0000

>> OMEG = 0:pi
OMEG =
    0   1.0000   2.0000   3.0000
```

2.2.3 Mathem. Funktionen und Operatoren für Vektoren und Matrizen

Anweisung	Beschreibung
`* ^ \`	Matrixoperationen: Multiplikation, Potenz, Linksdivision
`.* .^ ./`	Elementweise Operatoren
Matrix`.',` **`transpose(`***Matrix*`)`	Transponierte einer Matrix (Vektor ist 1D-Matrix)
Matrix`',` **`ctranspose(`***Matrix*`)`	Transponierte mit konjugiert-komplexen Elementen
`diff(`*Vektor [,n]*`)`	n-facher Differenzvektor, Default ist n=1.
`conv(`*Vektor1 , Vektor2*`)`	Faltung (convolution), z.B. Polynom-Multiplikation
f`([`*x1 x2 x3 ...*`])`	= `[`*f*`(`*x1*`) `*f*`(`*x2*`) `*f*`(`*x3*`) ...]`, elementweise Anwend.

Anweisung	Beschreibung		Anweisung	Beschreibung
`max(`*vec*`)`	größtes Vektorelement		**`inv`**(*matrix*)	Inverse einer Matrix: A^{-1}
`min(`*vec*`)`	kleinstes Element		**`det`**(*matrix*)	Determinante
`mean(`*vec*`)`	Mittelwert		e**`ig`**(*matrix*)	Eigenwerte
`std(`*vec*`)`	Standardabweichung		**`rank`**(*matrix*)	Rang
`sum(`*vec*`)`	Summe der Elemente		**`cumsum(`***vec*`)`	Kumulierte Summe
`prod(`*vec*`)`	Produkt der Elemente		**`cumprod(`***vec*`)`	Kumuliertes Produkt

Tabelle 2.2.3: Operatoren zur Verarbeitung von Vektoren und Matrizen.

`Matlab` steht nicht umsonst für **`Mat`**rix **`Lab`**oratory. Die Verarbeitung von Vektoren und Matrizen wird sehr genau genommen. Die Multiplikation zweier Vektoren ist in der Mathematik als Produkt von *Zeile* x *Spalte* definiert. Wenn $vec = (x, y, z)$, dann ist:

Falsch: $vec * vec = (x,y,z) \cdot (x,y,z)$, $vec' \cdot vec' = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix}$, richtig, $vec * vec' = (x,y,z) \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix}$.

Beispiel für Vektormultiplikation: x·x' = 1·2 + 2·2 + 3·3 = 14:

```
>> x = [1 2 3]
>> x * x'
ans =
     14
```

Operatoren mit einer Konstanten werden elementweise durchgeführt:

```
>> 4 * x - 1
ans =
     4     8    12
```

Es gilt weiterhin das Gesetz vor Punkt- vor Strichrechnung:

```
>> 4 * x - 1
ans =
     3     7    11
```

Operationen mit Funktionen werden elementweise durchgeführt:

```
>> x = -4*pi:0.01:4*pi;
>> y = sin(x);
>> plot(x,y);
```

Linksdivision: Gegeben seien Matrix **A**, Vektor **y**, **A·x=y** liefert für Vektor **x**=$\mathbf{A}^{-1}$**y=A\y**.

```
>> x = inv(A)*y;  % 1. Möglichkeit zur Berechnung von Vektor x
>> x = A \ y;     % 2. Möglichkeit zur Berechnung von Vektor x
```

Operatoren mit einem Punkt davor werden elementweise durchgeführt. Beispiel:

```
>> vec = 1:3
vec =
     1     2     3
>> v2 = vec.^2
v2 =
     1     4     9
>> si = sin(x) ./ x;
>> plot(x,si);
```

2.2.4 Strukturen

Hierarchieebenen von Datenstrukturen werden durch Punkte unterschieden. Beispiel:

```
>> Datum.Tag = 25; Datum.Monat = 'Mai'; Datum.Jahr = 2005;
>> Name.Vorname = 'Gerd';
>> Name.Zuname = 'Meier';
>> Person.Geburtsdatum = Datum;
>> Person.Name = Name;
>> Person(2) = Person;       % Person wird zum Vektor
>> Person(2).Geburtsdatum    % Abfragen Geb.-Dat. von Person 2
ans =
      Tag: 25
    Monat: 'Mai'
     Jahr: 2005
```

2.2.5 Cell Arrays

2.2.6 Verwalten von Variablen

Anweisung	Beschreibung
size(*Matrix***)**	Dimensionen einer Matrix oder Länge eines Vektors
length(*Matrix***)**	Größte Dim. einer Matrix oder Länge eines Vektors
clear	Löscht alle Veriablen im Workspace
clear all	Löscht zusätzlich alle globalen Veriablen
clear *[var1, var2, ...]*	Löscht die ausgewählten Veriablen
who	Liste aller Variablen im Workspace
whos	Detaillierte Liste aller Variablen im Workspace mit name, Dimension, Speicherbedarf und Datentyp

Tabelle 2.2.6: Operatoren zum Verwalten von Vektoren und Matrizen.

Beispiel für das Erfragen der Dimension von Vektoren und Matrizen:

```
>> mat=(ones(3,4,2))    % Matrix in drei Dimensionen
>> size(mat)
ans =
     3     4     2      % liefert Dimensionen
>> length(mat)
ans =
     4                  % liefert größte Dimension
```

Vektor-Indizes beginnen mit 1 (nicht mit 0!), der letzte Indes ist mit end abfragbar. Der Doppelpunkt steht für einen durchlaufenden Index. Man kann jede Matrix auch als Vektor indizieren, indem man nur einen Index verwendet, der spaltenweise abgearbeitet wird.

```
>> mat = [1 2 3; 4 5 6]
mat =
     1     2     3
     4     5     6
>> [mat(2), mat(5)]          % einfach indiziert: es wird
ans =                        % spaltenweise durchgezählt
     4     3
>> mat(end)                  % einfach indizierte Frage nach
ans =                        % letztem Element im Array
     6
>> mat(end,:)                % doppelt indiziert Frage nach
ans =                        % letzter Zeile im Array
     4     5     6
>> mat(:,end)                % doppelt indiziert Frage nach
ans =                        % letzter Spalte im Array
     3
     6
```

2.3 Ablaufsteuerung

2.3.1 Vergleichsoperatoren und logische Operatoren

Operator	Beschreibung
`==` , `eq(`*a*`,`*b*`)`	gleich
`~=` , `ne(`*a*`,`*b*`)`	ungleich
`<` , `lt (`*a*`,`*b*`)`	kleiner
`<=` , `le(`*a*`,`*b*`)`	kleiner oder gleich
`>` , `gt(`*a*`,`*b*`)`	größer
`>=` , `ge(`*a*`,`*b*`)`	größer oder gleich
`~` , `not(`*a*`)`	Negation
`&` , `and(`*a*`,`*b*`)`	UND
`&&`	Shortcut-UND (bricht ab sobald Ergebnis eindeutig)
`|` , `or(`*a*`,`*b*`)`	ODER
`||`	Shortcut-ODER (bricht ab sobald Ergebnis eindeutig)
`xor (`*a*`,`*b*`)`	XOR
`all(`*Vektor*`)`	=1 wenn jedes Vektorelement wahr
`any(`*Vektor*`)`	=1 wenn mindestens ein Vektorelement wahr
`logical(`*a*`)`	Typ-Umwandlung
`exist('`*x*`')`	Existenz von *x*
`find(`*Vektor*`)`	Indizes wahrer Elemente

Tabelle 2.3.1: Operatoren zum Verwalten von Vektoren und Matrizen.

Logische Operatoren können auf alle Zahlen angewendet werden. 0 ist logisch falsch, alles andere ist logisch wahr.

Reihenfolge der Auswertung: Potenz- vor Punkt- vor Strich- vor logischen Rechnungen:

```
>> help precedence
```

2.3.2 Verzweigungsbefehle `if` und `switch`

`if` *logischer_Ausdruck Anweisung[en]* [**`elseif`** *...]* [**`else`** *...]* **`end`**
`switch` *Ausdruck* **`case`** *Ausdruck Anweisungen* [**`case`** *...]* [**`otherwise`** *...]* **`end`**

Tabelle 2.3.2: Operatoren zum Verwalten von Vektoren und Matrizen.

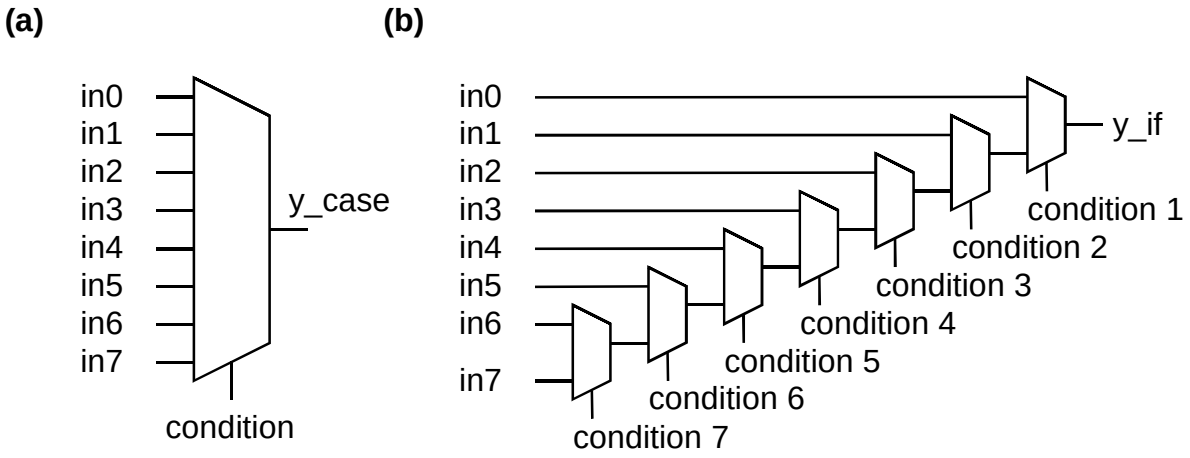

Bild 2.3.2: Unterschied zwischen **`IF`** und **`CASE`** Anweisungen:
(a) case: gleichwertige Eingänge, nur eine Bedingung, **(b) if:** Parity-bit coding.

Beispiel für **`if`**, die Zeilen unterscheiden sich in Komma oder Semikolon nach **`a=`**... :

```
>> x = 5;
>> if x<=2, a=2; elseif x>=3, a=5; else a=10; end   % a=...;
>> if x<=2, a=2, elseif x>=3, a=5, else a=10, end   % a=...,
a =
     5
```

Beispiel für **`switch`**, die Zeilen unterscheiden sich in Komma oder Semikolon nach **`a=`**... :

```
>> x = 5;
>> switch x case 2; a=2; case{3 4 5}; a=5; otherwise a=10; end
>> switch x case 2; a=2, case{3 4 5}; a=5, otherwise a=10, end
a =
     5
```

2.3.3 Schleifenbefehle `for` und `while`

`for` *variable = Vektor Anweisung[en]* **`end`**	
`while` *Ausdruck Anweisungen* **`end`**	

Tabelle 2.3.3: Operatoren zum Verwalten von Vektoren und Matrizen.

Beispiele einer **`for`** loop:

```
>> for k=0:2, k^2; end;
>> for k=0:2; k^2; end;
>> for k=0:2 k^2; end;
>> for k=0:2 k^2, end;
ans =
     0
ans =
     1
ans =
     4
```

Beispiel einer **`while`** loop:

```
>> k=0; while k<=2; k^2; k=k+1; end;
>> k=0; while k<=2, k^2; k=k+1; end;
>> k=0; while k<=2 k^2; k=k+1; end;
>> k=0; while k<=2 k^2, k=k+1; end;
ans =
     0
ans =
     1
ans =
     4
```

2.3.4 Abbruchbefehle `continue`, `break` und `return`

`continue`	innerhalb **`for`** oder **`while`** Schleife: sofort zum nächsten Iterationsschritt
`break`	innerhalb **`for`** oder **`while`** Schleife: Abbruch der Schleife
`return`	Bricht eine Matlab-Funktion oder -Skript ab

Tabelle 2.3.4: Operatoren zum Verwalten von Vektoren und Matrizen.

2.4 Matlab-Editor

Alle Matlab Anweisungen können auch in eine Datei mit der Endung ".m" geschrieben werden (sogenannte M-Files), die man dann ablaufen lässt. Der Aufruf erfolgt durch Schreiben des Dateinamens ohne die Extension ".m".

Editieren einer neuen M-Datei:

```
>> edit ... (→ save dateiname)
```

Wenn die Datei bereits existiert:

```
>> edit ... (→ save dateiname)
```

Aufruf einer M-Datei:

```
>> dateiname
```

Tabelle 2.4: Kommentar – und Fortsetzungzeilen

Zeichen	Beschreibung
`...`	Fortstezungzeichen zum Umbruch überlanger Zeilen
`%`	Beginn einer Kommentarzeile
`%{Kommentar %}`	Mehrzeiliger Kommentar
`%%`	Beginn eines Kommentars als Cell-Divider

Beispiel für eine Datei **`mittelwerte.m`**:

```
>> edit                          % Editieren einer neuen M-Datei
```

```
x = 1:10                              % x = [1 2 3 ... 10]
arithm = mean(x)                      % arithmetisches Mittel
geom   = prod(x).^(1/length(x))       % geometrisches Mittel
```

Speichern der Datei im Workspace unter "mittelwerte.m " und Aufruf in Matlab mit

```
>> mittelwerte     % Aufruf der Datei "mittelwerte.m"
```

2.5 Matlab-Funktionen

In M-Dateien können auch Matlab-Funktionen definiert werden gemäß der Schablone:

```
function [out1,out2,...] = function_name (in1,in2,...)
```

Tabelle 2.5: Schlüsselwörter, die innerhalb jeder Funktion abgefragt werden können.

Zeichen	Beschreibung
`nargin`	Anzahl der Eingagsargumente bzw. -variablen
`nargout`	Anzahl der Eingagsargumente bzw. -variablen
`nargchk(min,max,n)`	Anzahl der Übergabeparamtere überprüfen, ob gilt min ≤ n ≤ max, sonst Fehlertext
`isempty('name')`	Gültigkeit der Variablen name prüfen
`error('info')`	Abbruch der Funktion und Anzeige von info

Dabei sollte der Funktionsname gleich dem Dateinamen sein, denn aufrufen in Matlab muss man den Dateinamen, sonst findet er die M-Datei nicht.

Beispiel für eine Funktion in einer M-Datei:

```
>> edit ... (→ save f_mittelwerte.m)
```

```
function [arithm, geom] = f_mittelwerte(x)
arithm = mean(x);                    % arithmetisches Mittel
geom   = prod(x).^(1/length(x));     % geometrisches Mittel
```

Speichern der Datei im Workspace unter " `f_mittelwerte.m` " und Aufruf in Matlab mit

```
>> [a,g] = f_mittelwerte(x)
```

Da die Funktion zwei Werte zurückliefert, muss man ihr mit "`[a,g] =` " auch zwei Variablen anbieten, auf die sie schreiben kann. Bietet man zu wenig Variablen an, werden die letzten unterdrückt.

Beispiel: Hier wird nur die erste Variable zurückgegeben, da nur eine Zielvariable vorhanden:

```
>> a = f_mittelwerte(x)
```

Ein Semikolon hinter dem Funktionsaufruf unterdrückt die Ausgabe der Ergebnisse:

```
>> [a,g] = f_mittelwerte(x)
```

Auch die Zahl der Eingangsvariablen kann variieren. Wer von dieser Option gebrauch macht, sollte mit den Schlüsselwörtern nargin, nargout, nargchk innerhalb der Funktion die Zahl der aktuellen Ein- und Ausgangsparameter erfragen und kann mit `error('info')` Fehlermeldungen ausgeben.

2.5.5: Funktionen als Inline-Objekt:

Definition von Funktionen ohne M-File:

```
>> si = inline('sinc(t/pi)');
```

Die Inline-Funktion f1 kann nun weiter verwendet werden:

```
>> t = -10:0.1:10;
>> plot(t,si(t));
```

Definition von Funktionen ohne M-File:

```
>> f1 = inline('x.^2 + x - 1');
```

Die Inline-Funktion f1 kann nun weiter verwendet werden:

```
>> x = -10:0.1:10;
>> y = f1(x);
>> plot(x,y);
```

2.6 Code-Optimierung

2.7 Übungsaufgaben zum 2. Kapitel

3 Eingabe und Ausgabe in Matlab

3.1 Steuerung der Bildschirmausgabe

Die Befehle zur Steuerung der Bildschirmausgabe sind syntaktisch an UNIX (Linux) Befehle angelehnt.

Tabelle 3.1: Steuerung der Bildschirmausgabe

Anweisung	Funktion
`more` `more (n)`	Seitenweise Bildschirmausgabe (n Zeilen pro Seite), Return: zeilenweise Fortsetzung, _: Leertaste bewirkt seitenweise Fortsetzung, Q: Abbruch der Bildschirmausgabe
`echo [on|off]` `echo fname [on|off]`	Echo der abgearbeiteten Zeilen aus M-Files `[on|off]` Echo der abgearbeiteten Zeilen der Function `fname [on|off]`
`pause` `pause(n)`	Hält Bildschirmausgabe an Hält Bildschirmausgabe für n Sekunden an
`clc`	Löscht Ein- und Ausgaben im Comand-Window

3.2 Benutzerdialoge

3.2.1 Text in Matlab (Strings)

Strings in einfachen Hochkommata angeben, verketten mit [ ]. Zahlen sind mit der Funktion *num2str* unzuwandeln. Ausgabe auf den Bildschirm mit *disp(string)*.

Beispiel:

```
text = ['Dies ist ', 'ein ', ' Text!'];
```

3.2.2 Eingabedialog

Abfrage vom Bildschirm mit *variable=input(string)*. Wenn ein String eingegeben werden soll lautet die Abfrage *variable=input(string,'s')*. Als Sonderzeichen stehen der Zeilenumbruch, einzugeben mit \n, der Backslash \, einzugeben mit \\ und das Hochkomma ' , einzugeben mit '', zur Verfügung.

Abfrage-Beispiel:

```
order = input(['Was ist des Filter''s Ordnung?   \n', ...
               'PS: Das ist: Anzahl der Taps-1   ']);
ftyp  = input( 'Filtertyp [fir|iir] ?            ','s');
```

Im Matlab Command-Window sehen wir dann z.B.:

```
Was ist des Filter's Ordnung?
PS: Das ist: Anzahl der Taps-1   20
Filtertyp [fir|iir] ?            fir
```

3.2.3 Formatierte Ausgabe

Mit der Funktion *disp(string)* kann in das Matlab Command Window geschrieben werden.

Ausgabe-Beispiel (die erste Anweisung erzeugt eine Leerzeile):

```
disp(' ');
disp(['Filtertyp: ',ftyp]);
```

Mit der aus C bekannten Funktion `sprintf` kann formatiert auf einen String geschrieben werden. Statt doppelten Gänsefüßchen sind einfache Hochkommata zu verwenden.

Beispiel für Anweisungen:

```
ausgabe = sprintf('%s-Filter mit Ordnung %d.\n', ftyp, order);
disp(ausgabe);
```

Im Matlab Command-Window sehen wir dann z.B.:

```
fir-Filter mit Ordnung 20.
```

3.3 Import und Export von Daten

3.4 Betriebssystemaufruf und Dateiverwaltung

Matlab sucht M-Dateien oder aufgerufene Daten im aktuellen Verzeichnis (Default ist **`work`**) oder im Matlab-Pfad, der sich über das Menü

File/Set Path oder mit der Anweisung
`pathtool` einstellen lässt.

Tabelle 3.4: Anweisungen zur Dateiverwaltung

Anweisung	**Funktion**
`cd {`*Verzeichnis* **`.	..}`**	"change directory" in {Unter- | aktuelles | höreres} Verz.
`pwd`	Anzeige des aktuellen Verzeichisses
`dir [`*Auswahl***`]`** **`ls [`***Auswahl***`]`**	Hält Bildschirmausgabe an Hält Bildschirmausgabe für n Sekunden an
`mkdir` *Verzeichnis*	Löscht Ein- und Ausgaben im Comand-Window
`copyfile` *Quelle Ziel*	
`delete` *Datei*	
`!` *Kommando*	*Kommando* wird an das Betriebssytem weitergegeben
`eval(`*string***`)`**	*String* als Matlab-Anweisung interpretieren
`pathtool` oder Menü: File / Set Path	Öffnen des **Path Browsers**, solange er offen ist wird **`pwd`** nicht angenommen
`path` *Pfadname*	Der hier definierte Pfad wird neben dem aktuellen Verzeichnis nach M-Dateien und Daten durchsucht.

Beispiel für die Evaluierungs-Anweisung (Zeileninterpreter):

```
>> string = 'ls; cd ..; dir; cd .; ls; cd work; ls';
>> eval(string)
```

3.5 Grafische Darstellung

3.5.1 Die Figure – Grundlage einer Matlab-Grafik

Tabelle 3.5.1: Grafik allgemein

Anweisung	**Funktion**
`figure [(`*Nummer***`)]`**	Erzeugen (ansprechen) einer Figure
`subplot(`*Zeilen,Spalten,Zähler***`)`**	Erzeugen (ansprechen) einer Figure
`close [`*Nummer* **`| all ]`**	Figure *Nummer* bzw. alle Figures schließen (löschen)
`hold [off|on]`	Bei Neuplot alte Grafik in der Figure löschen | beibehalten
`clf`	Clear Figure
`gcf`	Get handle to Curret Figure
`gca`	Get handle to Curret Axis

Jede Grafik hat eine Nummer, die explizit oder automatisch vergeben werden kann und als Handle bezeichnet wird. Die Anweisung **`close [`***Nummer***`]`** schließt die Figure mit dem Handle Nummer; ohne Nummer wird die zuletzt angesprochene Figure geschlossen. Die zuletzt angesprochene Figure lässt sich mit gcf abfragen. Mit **`figure(3)`** wird die Figure mit dem Handle 3 erzeugt oder – falls sie bereits existiert – aktiviert. Eine existierende Figure lässt sich auch durch Anklicken mit der Maus aktivieren.

Die Anweisung **`subplot(3,2,6)`** unterteilt die aktuelle Figure in 6 Felder, die in 3 Zeilen und 2 Spalten angeordnet sind und spricht mit *Zaehler*=6 das 6. Feld (unten rechts) an. Es wird zeilenweise von links oben nach links unten durchnummeriert. *Zaehler* muss einen existierendem Subplot ansprechen, also $1 \leq Zaehler \leq Zeilen*Spalten$. Für ein Bodediagramm mit Betrag und Phase übereinander würde man also die Anweisung **`subplot(2,1,`***Zahler***`)`** verwenden.

3.5.2 Achsen und Beschriftung

Tabelle 3.5.2: Achsen und Beschriftung

Anweisung	**Funktion**
`axis('auto')`	Automatische Achsenskalierung
`axis([`*xmin*`,`*xmax*`,`*ymin*`,`*ymax*`])`**	Manuelle Achsenskalierung für 2D-Plot
`axis([`*x1*`,`*x2*`,`*y1*`,`*y2*`,`*z1*`,`*z2*`])`**;	Manuelle Achsenskalierung für 2D-Plot
`grid [off|on]`	Gitternetz aus | ein
`zoom [off|on]`	Zoom aus | ein
`xlabel(`*string*`)`**	Beschriftung der x-Achse
`ylabel(`*string*`)`**	Beschriftung der x-Achse
`zlabel(`*string*`)`**	Beschriftung der z-Achse
`title(`*string*`)`**	Überschrift
`text(`*x,y,string*`)`**	Text platzieren
`legend(`*sting1,...*`)`**	Legende

Die Skalierung der Achsen erfolgt automatisch oder nach **`axis('auto')`** wieder automatisch und kann mit axis abgefragt werden. Manuell kann sie für 2D-Plots mit **`axis([xmin,xmax,ymin,ymax])`** und für 3D-Plots mit der Anweisung **`axis([x1,x2,y1,y2,z1,z2])`**; die eckigen Klammern gehören dazu.

Gitterlinien lassen sich mit **`grid [off|on]`** eintragen.

Zoom kann man manuell mit **`zoom [off|on]`** oder den entsprechenden Knöpfen (Buttons) über der Grafik aktivieren und das zu zoomende Fenster mit der Maus definieren.

Mit **`xlabel(`***string*`)`**, **`ylabel(`***string*`)`**, **`zlabel(`***string*`)`**, lassen sich die entsprechenden Achsen beschriften, mit **`title(`***string*`)`** ein Titel erzeugen, mit **`text(`***x,y,string*`)`** ein Text platzieren und mit **`legend(`***sting1,..[,'Location',...]*`)`** eine Legende erzeugen und platzieren.

3.5.3 Festlegen von Farben, Markierungen und Linien in Plots

Tabelle 3.5.3: Festlegen von Farben, Markierungen und Linien in Plots

Farben		**Punkte**	**Linien**
k schwarz	r rot	. Punkte	- durchgezogen
b blau	m magenta	o Kreise	-- gestrichelt
c cyan	y gelb	* Sternchen	-. strich-punktiert
g grün	w weiß	+, x Kreuze	: gepunktet

Beispiele für die Anweisungen u Farben und Markierungen:

```
» x=[-3*pi:0.5:3*pi];
```

```
» y=sinc(x/pi);
» figure(6); plot(x,y,'+k-', x,sin(x),'m-.o', x,cos(x),'r*:');
» figure(7); plot(x,y,'k-+', x,sin(x),'mo-.', x,cos(x),'r:*');
```

Die Reihenfolge der Zeichen ist beliebig, daher zeigen Figur 6 und Figur 7 identische Plots.

3.5.4 Plot-Anweisungen für zweidimensionale Grafiken

Tabelle 3.5.4: Grafik: Importieren, Exorporiteren und Drucken

Anweisung	**Funktion**
plot(*[xwerte,] ywerte,... [,'Plotstil']***)**	Plot mit linearer Interplotaion der (x,y)-Paare. Default für x-Werte: Index der y-Werte. Mehrere Paare *xwerte, ywerte* sind erlaubt.
stairs(*[xwerte,] ywerte [,'Plotstil']***)**	Treppenfunktion, sonst wie **plot**
stem(*[xwerte,] ywerte [,'Plotstil']***)**	Taps (z.B. für DSP), sonst wie **plot**
bar(*[xwerte,] ywerte [,'Plotstil']***)**	Balkendiagramm, sonst wie **plot**
loglog(*xwerte, ywerte,... [,'Plotstil']***)**	Plot, beide Achsen logarithmiert
semilogx(*xwerte, ywerte,... [,'Plotstil']***)**	Plot, x-Achse logarithmiert
semilogy(*xwerte, ywerte,... [, 'Plotstil']***)**	Plot, y-Achse logarithmiert
polar(*Winkel, Radius,... [,'Plotstil']***)**	Plot, y-Achse logarithmiert
fplot(*Funktion,... [, Bereich]***)**	Plot einer expliziten Funktion
bode(*Funktion1 [,'Plotstil1'], ...***)**	Bode-Diagramm nach Betrag und Phase
ezplot(*Funktion1(x,y),... [, Bereich]***)**	

Beispiele für die Anweisungen **plot**, **stairs**, **stem**, **bar**:

```
» x=[-3*pi:0.5:3*pi];
» y=sin(x)./x;
» figure(1), plot(x,y);
» figure(2); plot(y);
» figure(3); stairs(x,y);
» figure(4); stem(x,y);
» figure(5); bar(x,y);
» figure(6); plot(x,y, x,sin(x)); grid on
```

Beispiel für die Anweisung **bode**: (Anwendung auf LTI-Modelle, → Kap. 5)

```
» s = tf('s'); % deklariere s zur Laplace-Variablen mit tf (transfer function)
» h01 = 1/(s^2 + 2*0.1*s + 1;
» h1  = 1/(s^2 + 2*1*s + 1);
» h10 = 1/(s^2 + 2*10*s + 1);
» figure(7); bode(h01, h1, h10);
```

Beispiel für die Anweisung **ezplot**:

```
» figure(8); ezplot('(x/2)^2 + (y/5)^2 = 1');
» hold on
» figure(8); ezplot('(x/2)^2 - (y/5)^2 = 1');
```

3.5.5 Plot-Anweisungen für dreidimensionale Grafiken

Tabelle 3.5.5: Grafik: Importieren, Exorporiteren und Drucken

Anweisung	**Funktion**
plot3(*xwerte, ywerte, zwerte... [,'Plotstil']***)**	3D-Plot, Punkte / Linien, mehrere Tripel *xwerte, ywerte , zwerte* sind erlaubt.
stem3(*xwerte, ywerte, zwerte... [,'Plotstil']***)**	Taps über der x/y-Ebene, sonst wie **plot**
bar3(*xwerte, ywerte, zwerte... [,'Plotstil']***)**	3-dimensionales Balkendiagramm
[X,Y] = meshgrid(*xvector, yvector***)**	Erzeugt Koordinatenmatrizen
surf(*xwerte, ywerte, zwerte... [,'Farbe']***)**	3D-Plot, Fläche (undurchsichtig)
mesh(*xwerte, ywerte, zwerte... [,'Farbe']***)**	3D-Plot, Gitter(durchsichtig)
waterfall(*xwerte, ywerte, zwerte... [...]***)**	3D-Plot, Wasserfall (Gradienten-Linien)
contour(*xwerte, ywerte, zwerte... [...]***)**	3D-Plot, Höhenlinien
box [off | on]	Plotumrandung einblenden
rotate3d [off | on]	Interaktives Drehen
view(*horizontal, vertical***)**	Perspektive ändern, [Default: -37.5°, 30°]
colormap(*name***)**	Wahl der Farbtabelle
caxis(*farbe_min, farbe_max***)**	Skalierung der Farbe

Die Anweisungen **plot3** und **stem3** wirken analog zu **plot** und **stem**, benötigen aber einen dritten Datenvektor für die z-Werte. Diese werden über der x/y-Ebene dargestellt.

Beispiele für die Anweisungen **plot3** und **stem3**:

```
» plot3(-sin(2*phi), cos(3*phi), 1.5*phi, 'r.-')
» grid on
» stem3(-sin(2*phi), cos(3*phi), 1.5*phi)
```

Beispiele für die Anweisungen **surf**, **mesh**, **waterfall**, **contour**:

```
» x=0:0.05:2; y=-1:0.1:1;
» [X,Y] = meshgrid(x,y)     % Erzeugt Koordinatenmatrizen
» z = (Y+1) .* cos(2*X.^2) + (Y-1) .* sin(2*X.^2)/5;
» figure
» subplot(221); surf(X,Y,Z);
» view(-40,30)
» subplot(222); mesh(X,Y,Z)
» subplot(222); waterfall(X,Y,Z)
» subplot(224); contour(X,Y,Z)
```

Nach Eingabe von **rotate3d** kann die Grafik mit der maus gedreht werden. Für große Grafiken sollte man Rotate Options / Plot Box Rotate wählen, damit die Grafik nicht während des Drehens neu gezeichnet wird.

3.5.6 Importieren, Exportieren und Drucken von Grafiken

Tabelle 3.5.6: Grafik: Importieren, Exorporiteren und Drucken

Anweisung	**Funktion**
`print -f`*nummer*	Figure auf Standraddrucker drucken
`print -f`*nummer* **`-d`***format 'Datei'*	Figure in Datei speichern
`saveas(`*handle, 'Datei' [,'Format']***`)`**	Als Matlab-Figure speichern, Default-Format ist *'fig'*
variable **`= imread(`***'Datei', 'Format'***`)`**	Pixel-Grafik in Matrix einlesen
`image(`*variable***`)`**	Pixel-Grafik (als Matrix gegeben) in Figure plotten

Beispiel:

```
>> my_photo = imread('photo.jpg', 'jpeg'); % liest Datei photo.jgp
>> image(my_photo);
>> saveas(1,'photo','fig'); % erzeugt Datei photo.fig
>> print -f1 %  Ausgabe der aktuelle Figure auf Standarddrucker (z.B. pdf)
```

Die unterstützten Formate können abgefragt werden, z.B. mit

```
>> more(30)   % immer nur 30 Zeilen auf einmal ausgeben
>> help imread
```

3.6 Grafische Benutzeroberfläche (GUI)

3.7 Tipps rund um die Matlab-Figure

3.8 Übungsaufgaben

4 Diffentialgleichungen in Matlab

5 Regelungstechn. Funktionen – Control System Toolbox

5.1 Modellierung von LTI-Systemen

Tabelle 5.1: Datentypen für LTI-Systeme:

Datentypen	**Beschreibung**
SISO	Single-Input / Single-Output
MIMO	Multiple-Input / Multiple -Output

5.1.1 LTI-Systeme

Tabelle 5.1.1: Darstellungarten von zeitkontinuierlichen LTI-Systemen:

Darstellungsart		**Beschreibung**	
TF	**tf(***num***,***den***)**	Transfer Function	Übertragungsfunktion
ZPK	**zpk(***z***,***p***,***k***)**	Zero-Pole-Gain	Pol-Nullstellen-Diagramm
SS	**ss(***a***,***b***,***c***,***d***)**	State Space	Zustandsdarstellung
FRG	**frd(***Ai***,***fi***,***eh***)**	Frequeny Response Data	Frequenzgang-Daten-Modelle zur spektralen Darstellung und Analyse von simulierten oder gemessenen Daten

Allgemeine Darstellung im Laplace-Bereich: $H(s) = \frac{num(s)}{den(s)} = \frac{a_m s^m + ... + a_1 s^1 + a_0}{b_n s^n + ... + b_1 s^1 + b_0}$

num: Numerator, Zähler, *den*: Denumerator, Nenner

Erstellen von TF-SISO-Systemen

```
>> num_vector = [am ... a1 a0]
>> den_vector = [bn ... b1 b0]
>> h = tf(num_vector, den_vector)
```

Beispiel: modelliert werden soll $h(s) = \frac{s+10}{s^2+2s+2}$

```
>> h = tf([1 10], [1 2 2])
```

Alternativ:

```
>> s = tf('s')          % s wird als Laplace-Variable interpretiert
Tansfer function
     s
>> h=(s+2)/(s^2+5*s + 4)
transfer function:
    s + 2
-------------
s^2 + 5 s + 4
```

Erstellen von TF-MIMO-Systemen

Beispiel 2 x 2: $H = \begin{bmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{bmatrix}$,

Matlab-Eingabe: h 0 [h11 h12 ; h21 h22]

5.1.2 Nullstellen-Polstellen-Darstellung - Zero-Pole-Gain ZPK

Jedes Polynom lßt sich in seine Nullstelln faktorisieren. Hat ein Polynom mt reellen Koeffizienten komplexe Nullstellen, dann treten diese als konjugiert-komplexe Paare $s_{z,1,2}=\sigma_z \pm j\omega_z$ auf.

Allgemeine Pol-Nullstellen-Darstellung in s: $H(s) = k \frac{(s-s_{z,1})\cdot(s-s_{z,2})\cdot\ldots\cdot(s-s_{z,m})}{(s-s_{p,1})\cdot(s-s_{p,2})\cdot\ldots\cdot(s-s_{p,n})}$

Beispiel: $H_{TF}(s) = \frac{3s-3}{s^2+11s+30} \quad \Leftrightarrow \quad H_{ZPK}(s) = 3\frac{s-1}{(s+5)(s+6)}$

Eingabemöglichkeiten der obigen Darstellungen in Matlab:

```
>> Htf=tf([3 -3], [1 11 30]);
>> Hzpk=zpk([1], [-5 -6], 3);
```

Alternativ: **s** als Laplace-Variable deklarieren und mit Operatoren (+, -, *, /, ^) verknüpfen:

```
>> s = tf('s')     % s als Laplace-Variable deklarieren
>> Hs1 = (3*s-3) / (s^2 + 11*s + 30)
>> Hs2 = 3*(s-1) / ((s+5)*(s+6))
```

5.1.3 Zustandsdarstellung – State-Space SS

5.1.4 Frequenzggang-Daten-Modelle – Frequency Response Data FRD

Eingabe diskreter Werte für Frequenzen f_i und zugehörigen Amplituden $A_i(f_i)$.

5.1.5 Zeitdiskrete Darstellung von LTI-Modellen

Verwendung der z-Transformation anstelle der Laplace (s-) Transformation.
Folge: Differenzengleichungen anstelle von Differentialgleichungen.
Zusätzlich benötigter Parameter: Abtastintervall $T_s = 1/f_s$ (sampling frequency)
Ist die Abtastezit noch nicht festgelegt, dann wird Ts=-1 gesetzt.

Tabelle 5.1.5: Darstellungarten für zeitdiskrete LTI-Systeme: zusätzliches Argument Ts

Darstellungsart		**Beschreibung**	
TF	**tf(***num***,***den[***,***Ts]***))**	Transfer Function	Übertragungsfunktion
ZPK	**zpk(***z***,***p***,***k [***,***Ts]***))**	Zero-Pole-Gain	Pol-Nulstellen-Diagramm
SS	**ss(***a***,***b***,***c***,***d[***,***Ts]***))**	State Space	Zustandsdarstellung
FRD	**frd(***Ai***,***fi***,***eh [***,***Ts]***)**	Frequeny Response Data	Frequenzgang-Daten-Modelle zur spektralen Darstellung und Analyse von simulierten o. gemessenen Daten
DSP	**filt**(*num***,** *den [***,** *Ts]*)	DSP-Format	Digital Signal Processing filter format

Allgemeine Darstellung im z-Bereich: $H(z) = \dfrac{num(z)}{den(z)} = \dfrac{a_m z^m + ... + a_1 z^1 + a_0}{b_n z^n + ... + b_1 z^1 + b_0}$

Allgemeine Pol-Nullstellen-Darstellung in z: $H(z) = k \dfrac{(z - z_{z,1}) \cdot (z - z_{z,2}) \cdot ... \cdot (z - z_{z,m})}{(z - z_{p,1}) \cdot (z - z_{p,2}) \cdot ... \cdot (z - z_{p,n})}$

Beispiel: $H_{TF}(z) = \dfrac{3z - 3}{z^2 + 1.1z + 0.3}$ = $H_{ZPK}(z) = 3 \dfrac{z - 1}{(z + 0.5)(z + 0.6)}$

Eingabemöglichkeiten der obigen Darstellungen in Matlab:

```
>> Htf  = tf([3 -3], [1 1.1 0.3], -1)
>> Hzpk = zpk([1], [-0.5 -0.6], 3, -1)
```

Alternativ: **z** als zeitdiskrete Variable deklarieren und mit Operatoren (+, -, *, /, ^) verknüpfen:

```
>> z = tf('z')    % s als zeitdiskrete Variable deklarieren
>> Hz1 = (3*z-3) / (z^2 + 1.1*z + 0.30)
>> Hz2 = 3*(z-1) / ((z+0.5)*(z+0.6))
```

Darstellung einer Übertragungsfunktion in z^{-1} (übliches Format für digitale Filter):

$$H(z) = \frac{num(z^{-1})}{den(z^{-1})} = \frac{a_0 + a_1 z^{-1} + ... + a_m z^{-m}}{b_0 + b_1 z^{-1} + ... + b_n z^{-n}} \qquad \left[= \frac{a_0 z^n + a_1 z^{n-1} + ... + a_m z^{n-m}}{b_0 z^n + b_1 z^{n-1} + ... + b_n} \right]$$

Beispiel: $H_{TF}(z) = \frac{3z - 3}{z^2 + 11z + 30}$ = $H_{DSP}(z) = \frac{3z^{-1} - 3^{-2}}{1 + 11z^{-1} + 30z^{-2}}$

5.1.6 Zeitverzögerungen in LTI-Modellen

5.2 Arbeiten mit LTI-Modellen

5.2.1 Eigenschaften von LTI-Modellen

Anweisung	**Beschreibung**
`get(`*sys***`)`**	Abfragen aller Modelleigeschaften von *sys*
sys.Eigenschaft	Abfragen der Modelleigeschaft *Eigenschaft* von *sys*
sys.Eigenschaft = Wert	Setzen der Modelleigeschaft *Eigenschaft* von *sys*

Tabelle 5.2.1: eigenschaften von LTI-Modellen

Ein LTI-Modell hat Eigenschaften, welche in Datenstrukturen abgelgt sind. Diese können mit der Anweisung

Eigenschaften eines LTI-Modelles abfragen und verändern:

Eingabemöglichkeit in Matlab:

```
>> Hzpk = zpk([1], [-0.5 -0.6], 3, -1)
>> get(Hdsp)
            num: {[0 3 -3]}
            den: {[1 11 30]}
       Variable: 'z^-1'
             Ts: -1
     InputDelay: 0
    OutputDelay: 0
  ioDelayMatrix: 0
      InputName: {''}
     OutputName: {''}
     InputGroup: {0x2 cell}
    OutputGroup: {0x2 cell}
          Notes: {}
       UserData: []

>> Hdsp.Notes
ans =
     {}

>> Hdsp.Notes = 'dies ist mein persoenlicher Kommentar';

>> Hdsp.Notes
ans =
    'dies ist mein persoenlicher Kommentar'
```

5.2.2 Schnelle Datenabfrage

Anweisung	Beschreibung
[*num*, *den*, *Ts***] = tfdata(***tf_sys***)**	Eigenschaften von *tf_sys* = tf(*num*,*den*,*Ts*)
[*z*, *p*, *k*, *Ts***] = zpkdata(***zpk_sys***)**	Eigenschaften von *zpk_sys* = zpk(*z*, *p*, *k*, *Ts*)
[*a*, *b*, *c*, *c*, *Ts***] = sskdata(***ss_sys***)**	Eigenschaften von *ss_sys* = ss(*a*, *b*, *c*, *c*, *Ts*)
[*Ai*, *fi*, *Ts***] = frddata(***frd_sys***)**	Eigenschaften von *frd_sys* = frd(*Ai*, *fi*, *Ts*)

Tabelle 5.2.2: Eigenschaften von LTI-Modellen auslesen

Diese Anweisungen sind ein wenig redundant, dann statt

```
>> [num, den, Ts] = tfdata(Htf);
```

kann man auch schreiben

```
>> num = Htf.num;
>> den = Htf.den;
>> Ts  = Htf.Ts;
```

5.2.3 Rangfolge der LTI-Modelle

Wenn wir die Schreibweise A > B definieren als „A hat Vorrang vor B“, dann gilt:

FRD > SS > ZPK > TF

5.2.4 Vererbung von LTI-Modelleigenschaften

Werden Modelle mit gleichen Eigenschaften verknüft (z.B multipliziert oder addiert), dann werden die gemeinsamen Eigenschaften auf das neue Modell übertragen: Zum Beispiel Funktion in **s**.

Vorsicht ist geboten, wenn Modelle mit verschiedenen Eigenschaften kombiniert werden, z.B. eine zeitkontinuierliche Funktion in **s** multipliziert wird mit einer zeitdiskreten Funktion in **z**. Oder wenn zeitdiskrete Fuktionen mit verschiedenen Tastraten verknüpft werden. Probleme diese Art werden hier nicht vertieft.

5.2.5 Umwandlung in einen anderen LTI-Modell-Typ

Zeitkontinuierliche Übertragungsfunktionen:

Beispiel: $H_{TF}(s) = \frac{3s + 3}{s^2 + 11s + 30}$ = $H_{ZPK}(s) = 3\frac{s+1}{(s+5)(s+6)}$

```
> % Erstellen von Übertragungsfunktionen mit tf und zpk:
> Htf=tf([3 3], [1 11 30])
> Hzpk=zpk([-1], [-5 -6], 3)

> % Umwandeln in den jeweils anderen Modelltyp:
> Htf_aus_Hzpk = tf(Hzpk)
> Hzpk_aus_Htf = zpk(Htf)
```

Zeitdiskrete Übertragungsfunktionen:

Beispiel: $H_{TF}(z) = \frac{3z+3}{z^2 + 0.11z + 0.30}$ $\Leftrightarrow$ $H_{ZPK}(z) = 3\frac{z+1}{(z+0.5)(z+0.6)}$

Eingabemöglichkeiten der obigen Darstellungen in Matlab:

```
> % Erstellen von Übertragungsfunktionen mit tf und zpk:
> Htf  = tf([3 3], [1 0.11 0.30], -1)
> Hzpk = zpk([-1], [-0.5 -0.6], 3, -1)

> % Umwandlen in den jeweils anderen Modelltyp:
> Htf_aus_Hzpk = tf(Hzpk)
> Hzpk_aus_Htf = zpk(Htf)
```

Umwandlungen zwischen zeitkontinuierlichen und -diskrete Übertragungsfunktionen:

Sie Unterkapitel 5.2.9: **`c2d`**, **`d2c`**, **`d2d`**.

5.2.6 Artihmetische Operationen

Arithmetische Operatioen wie +, -, *, /, ^ können auf LTI-Modelle angewendet werden. Verknüpft man verschiedene Arten von LTI-Modellen (z.B. ft und zpk) gilt die Rangfolge gemäß Unterkapitel 5.2.3.

```
>> s = tf('s')     % s als Laplace-Variable deklarieren
Transfer function:
s

>> Htf  = (3*s-3) / (s^2 + 11*s + 30)  % liefert tf-Modell
Transfer function:
    3 s - 3
---------------
s^2 + 11 s + 30

>> Hzpk = zpk([ ], [-5 -6], 3)      % zpk-Modell mit Nenner=1
Zero/pole/gain:
     3
-----------
(s+5) (s+6)

>> Hges = Htf * Hzpk                % liefert zpk-Modell
Zero/pole/gain:
    9 (s-1)
---------------
(s+6)^2 (s+5)^2
```

5.2.7 Auswählen, verändern und verknüpfen von LTI-Modellen

Anweisung	Beschreibung
parallel(*sys1***,** *sys2***,** *in1***,** *in2***,** *out1***,** *out2***)**	Parallelschaltung, *in1***=** *in2***,** *out=out1+out2*
series(*sys1***,** *sys2***,** *out1***,** *in2***)**	Serienschaltung: *out1=in2*
feedback(*sys_vorwaerts***,** *sys_rueckwaerts***)**	Rückg. System 1/(1+kA)

Tabelle 5.2.7: Verknüpfen von LTI-Modellen

Hier ist vor allem die Zusammenfügung einer Vorwärtsnetzwerkes A mit einem Rückkopplungsnetzwerk k zur NTF = A / (1+kA) interesant:

```
>> A = tf([1],[1 0])
Transfer function:
    1
    -
    s

>> k= 10;
>> NTF = feedback(A,k)
Transfer function:
      1
    ------
    s + 10
bode(NTF)
```

5.2.8 Spezielle LTI-Modelle

5.2.9 Umwandlung zwischen zeitkontin. und zeitdiskreten Systemen

Anweisung	Beschreibung
c2d(*sys_continuous***,** *Ts [***,** *Methode]***)**	zeitkontinuierliches in zeitdiskretes System umwandeln
d2c(*sys_discrete [***,** *Methode]***)**	zeitdiskretes in zeitkontinuierliches System umwandeln
d2d(*sys_ discrete***,** *Ts [***,** *Methode]***)**	zeitdiskretes in zeitdiskretes System umwandeln mit Änderung der Abtastzeit *Ts*
Methode	Diskretisierungsmethode: **'zoh'**, **'foh'**, **'tustin'**, **'imp'**, **'prewarp'**, **'matched'**

Tabelle 5.2.9: Umwandlungen zwischen zeitdiskreten / zeitkontinuierlichen LTI-Modellen

5.3 Analyse von LTI-Modellen

5.3.1 Allgemeine Eigenschaften

5.3.2 Modelldynamik

5.3.3 Systemantwort im Zeitbereich

Anweisung	Beschreibung
impulse(*sys [, t]* **)**	Impuls-Antwort berechnen
initial(*sys, x0, [t]* **)**	Anfangswert-Antwort berechnen
step(*sys [, t]* **)**	Sprung-Antwort berechnen
lsim(*sys, u, t [, x0]***)**	Systemantwort auf ein beliebiges Eingangsignal *u*
gensig(*typ, tau***)**	Testsignal generieren

Tabelle 5.3.3: Systemantwort im Zeitbereich

5.3.4 Systemantwort im Frequenzbereich

Anweisung	Beschreibung
evalfr(*sys, f* **)**	Berechnet Anwort bei einer komplexen Frequenz f
freqresp(*sys, omega***)**	Anworten für geg. Frequenzen *omega*·in rad/s
[*mag,ph,ω***] = bode(***sys [, omega]***)**	Bode-Diagramm, [Vorgabe der ω_i optional]
[*Gm, Pm, Wcg, Wcp***] = margin(***sys***)**	Amplitudenreserve *Gm*, Phasenreserve *Pm* und die Frequenzen, wo diese gem. werden
margin(*sys***)**	Grapfische Darstellung: *Gm, Pm, Wcg, Wcp*
allmargin(*sys***)**	Berechnet Stabilitätskenngrößen
nyquist(*sys [, omega]***)**	Berechnet das Nyquist-Diagramm

Tabelle 5.3.4: Systemantwort im Frequenzbereich

5.3.5 Interaktive Modellanalyse mit dem LTI-Viewer

5.3.6 Ordnungsreduzierte Darstellung

5.3.7 Zusatandsbeschreibungsform

5.4 Reglerentwurf

5.5 Probleme mit der numerischen Darstellung

5.6 Übungsaufgaben

6 Signalverarbeitung – Signal Processing Toolbox

6.1 Interpolation, Approximation und Abtastung

6.1.1 Interpolation und Approximation

6.1.2 Änderung der Abtastrate

Anweisung	Beschreibung
`downsample(`*Daten*`,` *Faktor [*`,` *Offset]* **`)`**	Reduktion von *fs* ohne Filterung
`upsample(`*Daten*`,` *Faktor [*`,` *Offset]* **`)`**	Erhöhung von *fs* ohne Filterung
`decimate(`*Daten*`,` *Faktor [*`,` *Ordnung] [*`,'fir'`*]***`)`**	Reduktion von *fs* mit Filterung
`interp(`*Daten*`,` *Faktor***`)`**	Erhöhung von *fs* mit Filterung
`resample(`*Daten*`,` *Zaehler*`,` *Nenner***`)`**	Änderung von *fs* mit Filterung

Tabelle 6.1.2: Operatoren zur Änderung der Abtastrate *fs = 1/Ts*.

6.2 Spektralanalyse

Digitale Fourier-Transformation (DFT) → Unterkapitel 6.2.4

6.2.1 Diskrete Fourier-Transformation (DFT)

6.2.2 Mittelung (Averaging)

6.2.3 Fensterung (Windowing)

Anweisung	Beschreibung
`rectwin(`*länge* **`)`**	Rechteck-Fenster (do-nothing window)
`triang(`*länge* **`)`**	Dreieck-Fenster, Randwerte > 0
`bartlet(`*länge* **`)`**	Dreieck -Fenster, Randwerte = 0
`hamming(`*länge* **`)`**	Hamming-Fenster, Randwerte > 0
`hann(`*länge* **`)`**	Hanning-Fenster, Randwerte > 0
`blackman(`*länge* **`)`**	Blackman-Fenster, Randwerte = 0
`wintool`	Windows Design & Analysis Tool aufrufen

Tabelle 6.2.3: Fensterfuktionen

6.2.4 Leistungsspektren

Anweisung	Beschreibung
`fft(`*x***`)`**	Diskrete Fourier-Transformation
`pwelch(`*x*`, `*fenst*`, `*ueb*`, `*Nfft*`, `*Fs*` )`**	Leistungsdichtespektrum
`cpsd(`*x*`, `*y*`, `*fenst*`, `*ueb*`, `*Nfft*`, `*Fs*`)`**	Kreuzleistungsdichtesprektrum
`specgram(`*x*`, `*Nfft*`, `*Fs*`)`**	Spektralanalyse nichtperiodischer Signale
`sptool`	Spgnal Processing Tool aufrufen

Tabelle 6.2.4: Funktionen zur Berechnung von Leistungsspektren

6.3 Korrelation

6.4 Analoge und Digitale Filter

6.4.1 Analoge Filter

Übertragungsfunktion: $H(s) = \frac{a_0 + a_1 s + ... + a_{m-1} s^{m-1} + a_m s^m}{b_0 + b_1 s + ... + b_{n-1} s^{n-1} + b_n s^n}$, Ordnung = max{m,n}

Anweisung	**Beschreibung**
besself(*Ordnung*, *omega***)**	Bessel-Filter (nur analog)
butter(*Ordnung*, *omega*,**'s')**	Butterworth-Filter
cheby1(*Ordnung*, *Welligkeit*, *omega*,**'s')**	Tschebyscheff-Filter: [z, p, k]
cheby2(*Ordnung*, *Welligkeit*, *omega*,**'s')**	Tschebyscheff-Filter : [A, B]
ellip(*Ordnung*, *Welligkeit*, *Daempfung*, *omega*,**'s')**	Elliptisches Filter (Cauer) /dB
[H, omega] = freqs(*Zaehler*, *Nenner [*,*omega]***)**	Frequenzgang analog

Tabelle 6.4.1: Erzeugung der Koeffizienten für analoge Filter.

Beispiel für einen Butterworth-Tiefpass 4. Ordnung, Erzeugung der Polynome A und B:

```
>> [A, B] = butter(4, 1000*2*pi, 's')   % TP 4. Ordnung, fg=1KHz
A =
  1.0e+015 *
  0       0       0       0       1.5585      %  a0 a1 a2 a3 a4
B =
  1.0e+015 *
  0.0000  0.0000  0.0000  0.0006  1.5585      %  b0 b1 b2 b3 b4
>> % Bodediagramm mit freqs( ) erzeugen:
>> freqs(A,B)                    % Bode-Diagramm plotten
>> % Bodediagramm "per Hand" erzeugen:
>> [H, omeg] = freqs(A,B);       % Übertragungsfunkt. und Frequenzen in rad/s
>> loglog(omeg/(2*pi), abs(H)) % Übertragungsfuntion plotten
```

Beispiel für einen Butterworth-Hochpass 4. Ordnung, Grenzfrequenz 1KHz; Bode-Plot:

```
>> [A, B] = butter(4, 1000*2*pi, 'high', 's'); freqs(A,B)
```

Beispiel für einen Butterworth-Bandpass 8. Ordnung, f_{g1}=1Hz, f_{g2}=1KHz; Bode-Plot:

```
>> [A, B] = butter(8, 2*pi*[1, 1000], 's'); freqs(A,B)
```

Beispiel für eine Butterworth-Bandsperre 8. Ordnung, f_{g1}=1Hz, f_{g2}=1KHz; Bode-Plot:

```
>> [A, B] = butter(8, 2*pi*[1, 1000], 'stop', 's'); freqs(A,B)
```

Beispiele für Tschbyschff-Tiefpass 6. Ordnung, Welligkeit dB, f_g=1KHz:

```
>> [A, B] = cheby2(6, 3, 2*pi*1000, 's'); freqs(A,B)
>> [z p k] = cheby1(6, 3, 2*pi*1000, 's') % Zeros-, Poles-Polyn., Gain
```

6.4.2 Digitale FIR – Filter

Übertragungsfunktion: $H(z^{-1}) = a_0 + a_1 z^{-1} + ... + a_{m-1} z^{-(m-1)} + a_m z^{-m}$, Ordnung = m

Der k-te Ausgangswert y(k) wird als gewichtete Summe des aktuellen Eingangswertes, x(k), und vergangener Eingangswerte, x(k-i), berechnet:

Zeitbereich: $y(k) = a_0 x(k) + a_1 x(k-1) + ... + a_{m-1} x(k-m+1) + a_m x(k-m)$

Anweisung	**Beschreibung**
fir1(*Ordnung***,** *normierte_Grenzfrequenz***)**	FIR Standard-Filter
fir2(*Ordnung***,** *normierte_Frequenzen***,** *Amplituden***)**	H(z): Polygonzug-Definition
filter(*Zaehler***, 1,** *Datenvektor***)**	diskrete Filterung von *Daten*
filtfilt(*Zaehler***, 1,** *Datenvektor***)**	diskr. Filt., phasenkorrigiert
[H, F] = freqz(*Zaehler***, 1,** *N***,** *Fs* **)**	Frequenzgang diskret

Tabelle 6.4.2: Erzeugung der Koeffizienten für digitale FIR - Filter. Alle Frequenzen werden normiert im Bereich 0...1 entsprechend 0...½f_s eingegeben.

Beispiele für **fir1**, Erzeugung des Polynoms A für einen Tiefpass 4. Ordnung, f_g=0,2·(½f_s):

```
>> A = fir1(4, 0.2)   % A = Koeffizienten für Tiefpass 4. Ordnung, fg=0,2·(½fs)
A =
  0.0284  0.2370  0.4692  0.2370  0.0284 % a0 a1 a2 a3 a4
>> A = fir1(40, 0.2, 'high');            % Hochpass, fg=0,2·(½fs)
>> A = fir1(40, [0.2, 0.4]);             % Bandpass, fg/(½fs)=0.2, 0.4
>> A = fir1(40, [0.2, 0.4], 'stop');     % Bandsperre, fg/(½fs)=0.2, 0.4

% Plotten des ensprechenden Passes:
>> [H, freq] = freqz(A, 1, 200, 100);
>> plot(freq, abs(H))
```

Als Default-Fenster wird das Hamming-Fenster verwendet. Einstellung der Rechteckfensters:

```
>> A = fir1(40, 0.2, rectwin(41)) % TP 40. O., fg=0,2·(½fs), do-nothing-win.
```

Vergleich der Wirkung der Fensterfunktionen im Frequenzbereich:

```
>> A1 = fir1(40, 0.2); [H1, freq1] = freqz(A1, 1, 200, 100);
>> A2 = fir1(40, 0.2, rectwin(41));
>> [H2, freq2] = freqz(A2, 1, 200, 100);
>> plot(freq1, abs(H1),freq2,abs(H2))
```

Vergleich der Wirkung der Fensterfunktionen im Frequenzbereich:

```
» freq = [0.0 0.3 0.3 0.6 1.0]; % Abszissen-Vektor normierter Frequenzen
>> amps = [1.0 1.0 0.5 0.5 0.0]; % Ordinaten-Vektor zugehöriger Amplitud.
>> A = fir2(50, freq, amps);
```

Mit der Anweisung "**filter(***Zaehler***, 1,** *Datenvektor***)**" wir das im Polynom A (und mit Nennerpolynom B=1) definierte digitale Filter auf die Daten im *Datenvektor* angewendet.

Wenn die zeitliche Verschiebung, die das Filter auf der Abszisse verursacht, unerwünscht ist, kamm man das Filter mit der der Anweisung "**filter(***Zaehler***, 1,** *Datenvektor***)**" einmal vorwärts und einmal rückwärts über die Daten laufen lassen und hat die zeitliche Verschiebung dadurch kompensiert. Allerdings hat sich die effektive Filterordnung auch verdoppelt.

Beispiel für die Anwendung von **filter** und **filtfilt**:

```
>> x = 1:100;                          % Vektor x als Absizze 1...100 erzeugen
>> Datenvektor = randn(1,100);         % 1 Zeile mit 100 Zufallszahlen (normalvert.)
>> A = fir1(20,0.2);                   % TP 20. Ordn. fg=0,2·(½fs), also Ts=10 Taps
>> y1 = filter(A,1,Datenvektor);       % Anwendung des TP, mit Verzögerung
>> y2 = filtfilt(A,1,Datenvektor);     % Anw. TP², Verzögerung kompensiert
>> plot(x,Datenvektor,x,y1,x,y2)       % grafische Darstellung von Dat., y1, y2
```

6.4.3 Digitale IIR – Filter

Übertragungsfunktion: $H(z^{-1}) = \frac{a_0 + a_1 z^{-1} + ... + a_{m-1} z^{-(m-1)} + a_m z^{-m}}{b_0 + b_1 z^{-1} + ... + b_{n-1} z^{-(n-1)} + b_n z^{-n}}$, Ordnung = max{m,n}

Der k-te Ausgangswert y(k) wird als gewichtete Summe des aktuellen Eingangswertes, x(k), vergangener Eingangswerte, x(k-i), und vergangener Ausgangswerte, y(k-i), berechnet:

Zeitbereich: $b_0 y(k) = a_0 x(k) + a_1 x(k-1) + ... + a_m x(k-m) - \left[b_1 y(k-1) + ... + y_n y(k-n)\right]$

Anweisung	**Beschreibung**
butter(*Ordnung***,** *normierte_Grenzfrequenz***)**	Butterworth-Filter
cheby1(*Ordnung***,** *Welligkeit***,** *normierte_Grenzfrequenz***)**	Tschebyscheff-Filter: [z, p, k]
cheby2(*Ordnung***,** *Welligkeit***,** *normierte_Grenzfrequenz***)**	Tschebyscheff-Filter : [A, B]
ellip(*Ordnung***,***Welligkeit***,***Daempfung***,***norm_Grenzfreq***)**	Elliptisches Filter (Cauer) /dB
yulewalk(*Ordnung***,** *normierte_Frequenzen***,** *Amplituden***)**	H(z): Polygonzug-Definition
filter(*Zaehler***,** *Nenner***,***Datenvektor***)**	diskrete Filterung von *Daten*
filtfilt(*Zaehler***,** *Nenner***,***Datenvektor***)**	diskr. Filt., phasenkorrigiert
[H, omega] = freqz(*Zaehler***,** *Nenner* **,***N* **,** *Fs***)**	Frequenzgang diskret
sptool	Signal Processing Tool
fdatool	Filter Design & Analysis Tool
fvtool(*Zaehler***,** *Nenner***)**	Filter Visualisation Tool

Tabelle 6.4.3: Erzeugung der Koeffizienten für analoge Filter.

6.4.4 Filterentwurf mit Prototyp - Tiefppässen

6.5 Übungsaufgaben

7 Optimierung – Optimization Toolbox

8 Simulink Grundlagen

8.1 Starten von Simulink

Matlab Command Window >> **`simulink`**

8.2 Erstellen und Editieren eines Signalflussplans

8.3 Simulations- und Parametersteuerung

8.4 Signale und Datenobjekte

Multiplexer

8.5 Signalerzeugung und –Ausgabe

8.5.1 Bibliothek: Sources - Signalerzeugung

cll → Sources

Source-Name	**Funktionebeschreibung** **Parameter; [Defaultwerte]**
Constant	Zeitunabhängiger reeller oder komplexer Wert Werte: Skalar, Vektor oder Matrix, [1]
Clock	Liefert die Simulationszeit Decimation [10] (Auslassungsfaktor), Display time [no]
Digital Clock	Liefert die Simulationszeit in Taktzyklen Sample time; [1]
Step	Sprungfunktion Step time, Initial value, Final Value, Sample time; [1 0 1 0]
Ramp	Rampe Parameter: Slope, Start Time, Initial Output; [1 0 0]
Signal Generator	Signal-Generator: Waveform, Amplitude, Frequency, Units, [sine 1 1 Hertz]
Pulse Generator	Puls-Generator: Period, Duty cycle, Amplitude, Start time, [1sec 50% 1 0]
Discrete Pulse Generator	Puls-Generator (ticked': in number of samples:) Amplitude, Period, Pulse width, Phase delay, Sample time, [1 2' 1' 0' 1]
Random Number	Gauß-verteilte Zufallszahlen, Mean, Variance, Initial seed, Sample time, [0 1 0 0]
Uniform Random Number	Gleichverteilte Zufallszahlen, Minimum, maximum, Initial seed, Sample time, [-1 1 0 0]
Band Limited White Noise	Bandbegrenztes und daher micht mehr ganz „weißes“ Rauschen Noise power, Sample time, Seed, [0.1 0.1 [23341]] Empfehlung aus Ref. [1]: Sample_time=1/(100τmin), mit τmin als kleinster Zeitkonstante im untersuchen System
Repeating Sequence	Wiederholte Sequenz, lineare Interpolation ansteigender Werte Time values, Output values, [[0 2] [0 2]]
Sine Wave	Sinuswelle Amplitude, Frequency, Phase/rad, Sample time, [1 1 0 0]
From Workspace	Daten aus dem Workspace lesen Data, Sample time; [[T,U] 0]
From File	Daten aus einer Datei lesen File name, Sample time [untitled.mat, 0]
Signal Builder	Verschiedene Signale über Multiplexer auswählbar

Tabelle 8.5.1: Signal-,Quellen in Simulink

8.5.2 Bibliothek: Sinks und Signal Logging

cll → Sinks

Möglichkeiten:

1. Grafische Darstellung,
2. Schreiben in Workspace
3. Schreiben in Datei

Sink-Name	**Funktionebeschreibung** **Parameter; [Defaultwerte]**
Skope	Oszilloskop: Number of axis [1], Time range [auto], Tick lables [bottom axis only], Decimation [1] (Auslassungsfaktor bei Tastung)
XY Graph	X-Y-Darstellung
Display	Numerische Darstellung
To File	
To Workspace	
Stop Simulation	

Tabelle 8.5.2: Signal-Senken in Simulink

8.5.3 Signal & Scope Manager

8.6 Mathematische Verknüpfungen und Operatoren

8.6.1 Bibliothek: Math Operations

Sum	
Product	'# of inputs' darf z.B. "*/" statt "2" sein
Dot Product	
Math Function	exp, log, sqrt, ...
Trigonometric Function	sin, cos, tan, ...
Gain	Skalierung mit Konstanten
Slider Gain	Verstärkung einstellbar mit Poti

8.6.2 Bibliothek: Logic and Bit Operations

8.7 Simulationsparameter

8.7.1 Die Configuration Parameters Dialog Box

8.7.2 Fehlerbehandlung und Fehlersuche

8.8 Verwaltung und Organisation eines Simulink-Modells

8.8.1 Arbeiten mit Callback-Routinen

Initialisieren von Simulink-Funktionen mit M-Files

8.8.2 Der Model-Browser

Navigieren in komplexen, hierarchischen Simulink-Modellen

8.8.3 Bibliotheken: Signal Routing und Singal Attributes

8.8.4 Drucken und Exportieren eines Simulink-Modells

8.9 Hierarchiebildung

8.10 Übungsaufgaben

9 Lineare und nichtlineare Systeme in Simulink

9.1 Bibliothek: Continuous – Zeikontinuierliche Systeme

9.2 Analyse eines Simulink-Modells

9.3 Bibliothek: Discontinuies – Nichtlineare Systeme

9.4 Bibliothek: Lookup-Tables – Nachschlagetabellen

9.5 Bibliothek: User-Defined Functions

9.6 Algebraische Schleifen

9.7 S-Funktionen

9.8 Übungsaufgaben

10 Abtastsysteme in Simulink

10.1 Allgemeines

10.2 Simulationsparamter

10.2.1 Rein zeitdiskrete Systeme

10.2.2 Hybride Systeme (gemischt zeitdiskret und zeitkontinuierlich)

10.3 Simulink-Bibliothek: Discrete – Zeitdiskrete Systeme

10.4 Gemischte Abtastzeiten und hybride Systeme

10.4.1 Gemischte Abtastzeiten

10.4.2 Hybride Systeme

10.5 Übungsaufgaben

11 Regelkreise in Simulink

11.1 Die Gleichstrom-Nebenschluss-Maschine GNM

11.2 Untersuchung der Systemeigenschaften

11.3 Kaskadenregelung

11.4 Zustandsbeobachter

11.5 Zustandsregelung mit Zustandsbeobachter

11.6 Initialisierungsdateien

11.7 Übungsaufgaben

12 Stateflow

Stateflow ist eine grafische Erweiterung von Simulink zur Modellierung und Simulation endlicher zustandsmaschinen (Finite State Machines)

12.1 Elemente von Stateflow

12.2 Strukturierung und Hierarchiebildung

12.3 Action Language

12.4 Anwendungsbeispiel: Getränkeautomat

12.5 Anwendungsbeispiel: Heizgebläse

12.6 Übungsaufgaben

13 Quellen

[1] A. Angermann, M. Beuschel, M. Rau, U. Wohlfarth: „Matlab – Simulink – Stateflow, Grundlagen, Toolboxen, Beispiele“, Oldenbourg Verlag, ISBN 3-486-57719-0, 4. Auflage (für Matlab Version 7.0.1, Release 14 mit Service Pack 1). URL: http://www.matlabbuch.de/