Technische Fachhochschule Belin
Fachbereich VI (Informatik und Medien)

Manfred Ottens

Einführung in das
CAE-Programm M A T L A B - SIMULINK
(Version 5.3)

Berlin, Sommersemester 2008

Inhaltsverzeichnis

1 Einleitung

MATLAB (MATrix-LABoratory) ist ein sehr hoch entwickeltes Computer-Aided-Engineering-Programm zur numerischen Lösung mathematischer Problemstellungen unter anderem aus den folgen Gebieten:

Mathematik: Lineare Algebra, Differentialgleichungen, Matrixoperationen, Approximation, Interpolation, nichtlineare und lineare Optimierung.

Systemtheorie: Lösung aller wichtigen Problemstellungen von MIMO-LTI-Systemen (MIMO: Multiple Input / Multiple Output-Systems; LTI: Linear-Time-Invariant-Systems) Systemgenerierung, Systemanalyse im Zeit-, Bild- und Frequenzbereich. Systemsimulation. Grundzüge nichtlinearer Systeme.

Regelungstechnik: Analyse und Synthese von Regelkreisen im Zeit-, Bild- und Frequenzbereich, inklusive der modernen Methoden des Zustandsraums; sowohl für kontinuierliche als auch für zeitdiskrete Systeme.

Systemidentifikation: Bildung von parametrischen mathematischen Modellen linearer Systeme auf der Basis gemessener Eingangs-/Ausgangsdaten des Systems (Parameterschätzverfahren und spektralanalytische Methoden).

Signalverarbeitung: Fouriertransformation (Spektralanalyse), moderne Entwurfsverfahren für digitale und analoge Filter (FIR- und IIR-Filter)

Optimierung: Lösung von linearen und nichtlinearen Optimierungsproblemen mit Randbedingungen (suchstrategische Optimierung), Entwurf optimaler Zustandsregler.

(Andere Anwendungsgebiete siehe /5/)

Matlab hat sich wegen seiner leichten Anwendbarkeit, seines außerordentlichen Leistungsumfangs, seiner Rechengenauigkeit und Schnelligkeit als Handwerkszeug für die Lösung numerischer Problemstellungen in nahezu allen Wissenschaftsbereichen weltweit etabliert.
Im Gegensatz zu anderen numerischen Rechen- und Simulationsprogrammen kann Matlab mit einer Programmerweiterung, dem "Real Time Workshop", über geeignete A/D-D/A-Wandler-Hardware auch auf die reale Umwelt zugreifen. Diese sogenannten "Rapid Prototyping-" oder auch "Hardware on the Loop-" Fähigkeiten machen Matlab für die Natur- und Ingenieurwissenschaften noch wertvoller.
Matlab ist grundsätzlich einfacher zu programmieren als BASIC, wenn man einmal verstanden hat, daß die Variablen in Matlab keine einfachen Zahlen (Skalare) sondern Zahlenfelder sind.
Matlab besteht aus einem Grundprogramm, nämlich **MATLAB** selbst und einer Reihe von sogenannten Toolboxes, die den Grundbefehls-Vorrat um spezielle Befehle für

bestimmte Wissenschaftsgebiete (z.B. Regelungstechnik, Kartographie, Finanzwissenschaften, symbolische Mathematik) erweitern.
Die für die Signal- und Systemwissenschaften wichtigste Matlab-Erweiterung ist **SIMULINK**, eine Programm-Erweiterung zur grafikgestützten Synthese und Simulation von vermaschten Systemstrukturen. Simulink kann wiederum durch sogenannte Blocksets für spezifische Wissenschaftsgebiete (z.B. Digital-Signal-Processing- (DSP-), Neural-Network-, Power-System-Blockset) erweitert werden.
Die dritte Matlab-Erweiterung ist **STATEFLOW**, ein grafikgestütztes Simulations-system für ereignisgesteuerte Zustandsautomaten.

In dieser Einführung werden neben dem Grundbefehlssatz von Matlab auch einige Befehle aus der Control System Toolbox und Simulink (ohne Erweiterungs-Blocksets) verwendet und erläutert. Alle Erläuterungen beziehen sich auf die folgenden Versionen:

- Matlab Version 5.3
- Control System Toolbox Version 4.2
- Simulink Version 3.0 .

Es existieren bereits neuere Matlab-Versionen. Matlab ist jedoch weitgehend aufwärtskompatibel, so daß die meisten Ausführungen dieses Skriptes z.B. auch für höhere Matlab-Versionen gelten.

In diesem Skript kann nur ein kleiner Ausschnitt der Leistungsfähigkeit von Matlab dargestellt werden. Dabei wird sich an den Lehrinhalten von /1/ und /2/ orientiert, mit dem Ziel, daß alle dort beschriebenen Methoden und Verfahren numerisch mit Matlab bzw. Simulink nachempfunden werden können.
Auch die Funktionalität der einzelnen Matlab-Befehle kann aus Zeit- und Platzgründen nicht vollständig beschrieben werden. In den meisten Fällen können die Befehle mit anderen Parametrierungen andere Leistungen erbringen. Diese Eigenschaften können jedoch einfach mit dem noch zu besprechenden Befehl "help" in der Matlab-Arbeitsumgebung auf dem PC nachgelesen werden.
Auch ein gelegentliches Nachschlagen einzelner Befehle in den Matlab-Handbüchern /3/ ist empfehlenswert

Auch die Nutzung der integrierten Entwicklungsumgebung (Editor/Debugger) wird nur so weit beschrieben, wie sie zu einer halbwegs professionellen Nutzung von Matlab notwendig ist.

1.1 Ein einführendes Beispiel

Zur Demonstration der Leistungsfähigkeit von Matlab soll das folgende kurze Programm "matl_dem.m" (Matlab Demonstration) dienen, das die Nullstellen und Extrema eines Polynoms 5.Ordnung berechnet und die Graphen des Polynoms y=f(x) und seiner Ableitung y'=f'(x) darstellt. (Die Funktion des Programms muß an dieser Stelle noch nicht vollständig begriffen werden.)

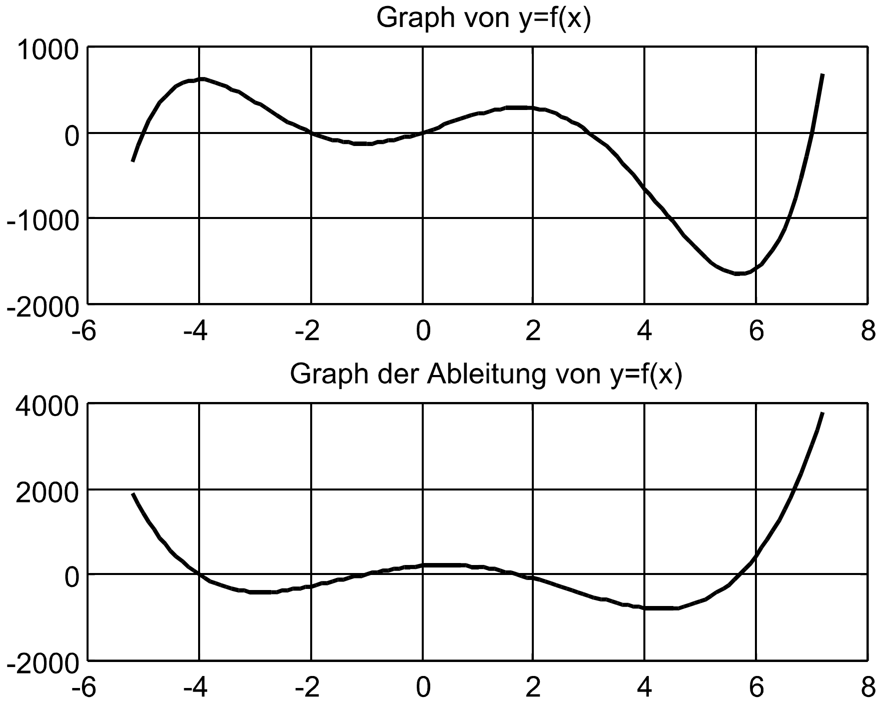
Die folgende Kode-Sequenz, ein sogenanntes Matlab Script-File

```
% File "matl_dem.m"
%
% Berechnung der Nullstellen, der Extrema und der Graphen
% von y=f(x) und der Ableitung y'=f'(x) eines Polynoms 5.Ordnung:
%
%    y=x^5-3x^4-39x^3+47x^2+210x  .
%

home                            % Säuberung des Command Windows.
clear

x=-5.2:0.1:7.2;                 % Betrachteter Abszissenbereich.
y_koeff=[1,-3,-39,47,210,0];    % Polynomkoeffizienten.
y=polyval(y_koeff,x);      % Berechnung des Graphen von y=f(x).
subplot(2,1,1)             % Graph von y(x) in die obere
plot(x,y), grid            % Bildhälfte.
title('Graph von y=f(x)')
Nullstellen=roots(y_koeff) % Berechnung und numerische
                           % Ausgabe der Nullstellen von f(x).
dy_dx_koeff=polyder(y_koeff);
                           % Berechnung der Koeffizienten von f'(x).
dy_dx=polyval(dy_dx_koeff,x);
                           % Berechnung von des Graphen von f'(x).
subplot(2,1,2)             % Graph von f'(x) in die untere
plot(x,dy_dx), grid        % Bildhälfte.
title('Graph der Ableitung von y=f(x)')
Extrema=roots(dy_dx_koeff) % Berechnung und numerische
                           % Ausgabe der Nullstellen von f'(x),
                           % (= Extrema von f(x)).
```

gibt auf dem Matlab Command Window folgende alpanumerischen Werte

```
Nullstellen =          Extrema =

     0                  5.6999
     7.0000            -3.9878
    -5.0000             1.7461
     3.0000            -1.0582
    -2.0000
```

und auf einem separaten Grafik-Bildschirm folgende Grafik aus:



Wie man deutlich erkennt, wird die anspruchsvolle Aufgabenstellung mit nur wenigen Kodezeilen gelöst. Dies wird durch spezielle leistungsfähige Befehle, wie z.B. "plot" (selbstskalierender Grafikausgabe-Befehl) oder "roots" (Befehl zur Berechnung der Nullstellen von Polynomen beliebiger Ordnung) ermöglicht. Auch das Fehlen jeglicher Programmschleifen fällt auf. Dies ist auf die Zahlenfeldorientierung von Matlab zurückzuführen, die wir in den nächsten Kapiteln kennenlernen werden.

1.2 Schriftkonventionen

Im folgenden Text werden die Matlab-Befehle **fett** gedruckt, z.B.:

[betrag,phase] = **bode**(zaehler, nenner, omega_bereich);.

Obwohl auch die Klammern, Kommas und Gleichheitszeichen vorgeschriebene Syntaxelemente sind,

[betrag,phase**] = bode(**zaehler**,** nenner**,** omega_bereich**);**

wird auf deren fette Hervorhebung verzichtet. Die frei wählbaren Variablennamen, und die Funktionsparameter der Matlabbefehle (z.B. betrag oder nenner) werden in normaler Schriftstärke dargestellt.

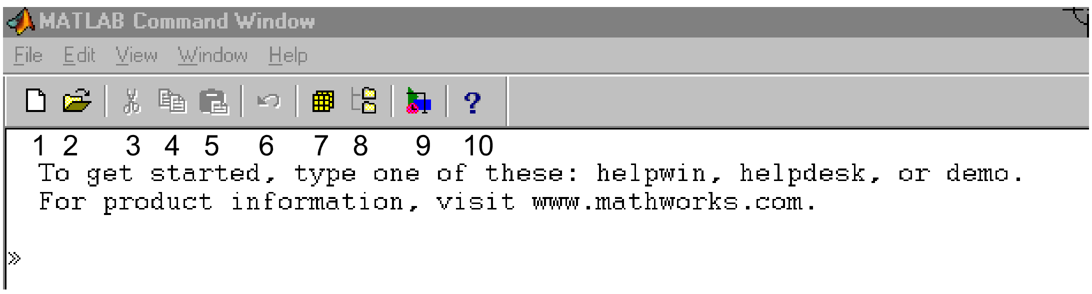
1.3 Inhalte des Skripts

Dieses Skript führt im **Kapitel 2** zunächst in die *Grundlagen der Matlab-Programmierung* ein, dazu gehören eine Beschreibung der Arbeitsumgebung auf dem Desktop, nämlich das Matlab-Command-Window, der Matlab-Editor, die Matlab-Hilfe, usw. Zu den Grundlagen gehören weiter Variablen und Datentypen, grundlegende mathematische Funktionen, Datenstrukturen, Eigenschaften der Datenhaltung, die Programmlaufkontrolle und die sogenannten Matlab m-Files.
Im **Kapitel 3** werden *Matlab-Funktionen für Problemstellungen aus der Systemtheorie und der Regelungstechnik* besprochen. Dazu gehören Methoden zur Generierung kontinuierlicher und zeitdiskreter Systemmodelle, ihrer Wandlung in verschiedene Betrachtungsbereiche (Zeitbereich, Bildbereich) und verschiedene Darstellungsformen (Polynom-, Produkt-, V-Normal,- Partialbruchform der Übertragungsfunktion) und der Systemanalyse (Systemantwortberechnung, Bodediagramme). Im **Kapitel 4** wird das *grafikgestützte Simulationssystem SIMULINK* eingeführt. Hier werden sehr ausführlich die Arbeitsumgebung erläutert, die verschiedenen grafischen Simulationselemente eingeführt und die wichtigen Schnittstellen zwischen Matlab und Simulink besprochen.

2. Grundlagen von Matlab

2.1 Aufruf der Arbeitsumgebung

Matlab läuft unter dem Betriebssystem Windows ab Version 95. Durch einen Doppelklick auf das Matlab-Icon auf dem Windows-Desktop gelangt man in das <u>Matlab Command Window</u> (MCW) (siehe Fensterausschnitt):



Das Matlab-Command-Window (MCW)

Am linken oberen Bildrand befinden sich verschiedene Pulldown-Menüs, deren Funktion mit denen von Windows weitgehend identisch ist. Von den darunter-liegenden Pushbuttons wiederholen die ersten sechs Funktionen des Pulldown-Menüs. Die Buttons 1, 7, 8 und 9 haben matlab-spezifische Bedeutung:

Button **1**: öffnet den Matlab-Editor/Debugger
Button **7**: öffnet den Workspace Browser
Button **8**: öffnet den Path Browser
Button **9**: öffnet die Simulink-Library

Da uns an dieser Stelle wesentliche Kenntnisse über Matlab fehlen, können diese Funktionen erst später erläutert werden. Wir gehen im Kapitel 2.7 intensiv darauf ein.

2.1.1 Hilfefunktionen

Ein Element der Matlab-Arbeitsumgebung soll allerdings schon an dieser Stelle eingeführt werden: Matlab besitzt umfangreiche **Hilfefunktionen**, die in drei Kategorien eingeteilt werden können und durch die Anweisungen

help, **helpwin** und **helpdesk** (jeweils mit CR abschließen)

im Matlab-Command-Window aufgerufen werden. Die ausführlichste Hilfe liefert **helpdesk**. Sie ist besonders für Einsteiger in Matlab geeignet. **helpwin** hilft dem erfahreneren Matlab-Programmierer bei der Suche nach Befehlen und gibt Hinweise zur Parametrierung von Befehlen. **help** allein hat eine ähnliche Funktion wie **helpwin**, alle Tätigkeiten spielen sich dann aber ausschließlich im MCW ab.

helpdesk:
Nach Eingabe der Anweisung ***helpdesk*** öffnet sich ein Internet-Explorer-Fenster in dem in zwei Spalten ***Matlab topics*** und ***Other products*** verschiedene Links angeboten werden. Für den Einsteiger ist die linke Spalte von Wichtigkeit. In den verschiedenen Links wird etwas mehr Stoff erläutert als in diesem Skript. Der Link ***Dokumentation Roadmap*** gibt Hinweise, wie man bei der Durcharbeitung des **helpdesk**s vorgehen kann. Der Link ***Matlab Functions by Index*** führt auf eine intensive Erläuterung der einzelnen Matlab-Befehle. Diese sind hervorragend zu gebrauchen, wenn man die Befehlsnamen kennt. Der Link ***Matlab Functions by Subjek****t* führt über übergeordnete Begriffe zur Erläuterung einzelner Befehle. ***Using Matlab*** enthält eine sehr ausführliche Beschreibung der Arbeitsumgebung und der Matlab-Funktionalität. Die Inhalte sind nach Kapiteln geordnet. Diese sind zur Stoffvertiefung hervorragend geeignet.
Die zweite Hauptspalte ***Other products*** kann später zur vertiefenden Erlernung des Umgangs mit ***Simulink*** benutzt werden.

helpwin:
Nach Eingabe der Anweisung **helpwin** öffnet sich ein Windows-Fenster in dem von Matlab und allen installierten Toolboxes und Blocksets Oberbegriffe (topics) von Befehlsinhalten ausgegeben werden. Durch einen Doppelklick auf einen solchen Oberbegriff erfolgt ein Sprung in die entsprechende Befehlsliste, die wiederum in Kategorien eingeteilt ist. Nach einem Klick auf den entsprechenden gesuchten Befehl erhält man dessen Beschreibung.

help:
Durch Eingabe von **help** im MCW werden auch hier von allen installierten Toolboxes und Matlab selbst Oberbegriffe (topics) von Befehlsinhalten ausgegeben Mit dem Befehl **help Oberbegriff** werden die vorhandenen Befehle dieser Gruppe mit ihrem Befehlsnamen genannt und kurz erläutert. Gibt man nun den Befehl **help Befehlsname** ein, wird schließlich der Befehl selbst erklärt.

Die Anweisung **lookfor Zeichenfolge** durchsucht die erste Kommentarzeile jedes Matlab-Befehls oder Matlabprogramms nach dem String "Zeichenfolge" und gibt bei Erfolg diese Zeile mit dem Befehlsnamen aus. Damit kann man sich wiederum mit **help Befehlsname** die Funktion des Befehls erläutern lassen.
Mit der Anweisung **which Befehlsname** kann festgestellt werden, aus welcher Toolbox der Befehlsname stammt.

2.2 Programmiergrundlagen

Nach dem Anklicken des Matlab-Icons öffnet sich das Matlab-Command-Window, auf dem zunächst ein Hinweistext ausgegeben wird. In der nächsten Zeile gibt Matlab das Zeichen **>>** aus und wartet auf eine Eingabe. Matlab besitzt auf dieser Ebene einen Zeileneditor, mit dem kleine, einfache Rechenoperationen und kleine Programmierarbeiten durchgeführt werden können.

2.2.1 Variablen und Datentypen

Allgemein gelten für Matlab folgende Grundregeln:

- Matlab ist eine Ausdruckssprache, die auf der Basis von Zuweisungen arbeitet (Variable = Ausdruck):
  >> a = 3 * 7 (CR) (erzeugt a = 21).

  Wird nur ein Ausdruck eingegeben, erzeugt Matlab die Variable "ans" (answer)
  >> 2 * 7 (CR) (erzeugt ans = 14).

Alle Eingaben im MCW werden mit der Betätigung der Eingabetaste **(CR)** abgeschlossen.

- Zuweisungs-Anweisungen können mit oder ohne Semikolon abgeschlossen werden (a = 7; oder a=7):
  Mit Semikolon: der Ausdruck wird berechnet, der Variablen zugewiesen aber nicht auf dem Bildschirm ausgegeben.
  Ohne Semikolon: wie mit Semikolon, aber das Berechnungsergebnis wird auf dem Bildschirm ausgegeben.

- Werden Variablen längere Ausdrücke zugeordnet, kann der Ausdruck mit drei Punkten auf die nächsten Zeile verlängert werden:

```
s = 1 - 1/2 + 1/3 - 1/4 ...
+1/5 - 1/6                    (erzeugt s = 0.6167)
```

- Datentypen und -Strukturen: Matlab kennt eine Reihe von Datentypen und Datenstrukturen (nähere Einzelheiten: **help datatypes**). Dennoch kommen Matlab-Programme auch ohne Variablendeklarationen aus. Dies ist so, weil jeder Ausdruck, der einer Variablen zugewiesen wird zunächst allgemein als

Zahlenfeld (Array) mit Zahlen des Typs "double"

aufgefaßt wird.
Die anderen noch existierenden Datentypen (z.B. int8, int16, single) müssen per Befehl aus double-Typen gewandelt werden. Mit den daraus entstehenden Datentypen können allerdings keine Rechenoperationen durchgeführt werden. Sie dienen lediglich bei der Handhabung großer Datenmengen (wie sie z.B. bei der Bildverarbeitung auftreten) der Speicherplatzoptimierung und Verringerung der Zugriffszeit bei Speicheroperationen. Um sie rechnerisch verknüpfen zu können, müssen sie wieder per Befehl in double-Typen gewandelt werden.
Wir werden in dieser Einführung außer dem Typ "double" nur noch den Datentyp "charakter" (char) und den daraus abgeleiteten Charakter-String nutzen. Die Typendeklaration erfolgt implizit an einer beliebigen Stelle im Programm durch Einrahmung des Ausdrucks im Hochkommas:

```
a = 'j';
antwort = 'nein';
```

Da, wie auch schon bemerkt, eine Variable grundsätzlich als Zahlenfeld (Array) aufgefaßt wird, muß diese Datenstruktur auch nicht deklariert werden. Auch eine Dimensionierung des Arrays ist nicht nötig. Es gibt jedoch zwei weitere

Datenstrukturen, sogenannte *Structures* und *Cell-Arrays*, die explizit generiert werden müssen (siehe Kapitel 2.2 4).

- Zahlenbereich, Auflösung: Wie vorangehend bemerkt, werden Matlab-Rechenoperationen ausschließlich mit dem Datentyp "double" durchgeführt. Dabei kann damit einen Zahlenbereich von ca. 10^{-300} bis 10^{300} mit einer Mantissenauflösung von ca. 10^{-16} überstrichen werden.

- Zahlen können im Dezimalformat oder im Mantissen-/Exponenten-Format eingegeben werden. Alle Eigaben mit den folgenden Formaten sind identisch und erlaubt. Dabei steht das "e" für 10^, d.h. als Zehnerpotenz mit dem dahinter stehenden Exponenten:

  0.314 = .314 = 314.0e-3 = 0.0314e1.

  Alle Berechnungen werden unabhängig von Eingabeformat mit der oben angegebenen Auflösung durchgeführt. Der Befehl **format** wird benutzt, um zwischen verschiedenen <u>Ausgabe</u>formaten zu wählen:

format :	Grundeinstellung, sonst wie short
format short :	Festpunktformat mit fünf Stellen
format long :	Festpunktformat mit 15 Stellen.
format short e :	Fließkommaformat mit fünf Stellen.
format long e :	Fließkommaformat mit 15 Stellen.
format rat :	Bruchapproximation mit kleinen ganzen Zahlen.

- Variablennamen dürfen beliebig lang sein, ausgewertet werden allerdings "nur" die ersten 31 Zeichen. Am Anfang muß ein Buchstabe, dann dürfen weitere Buchstaben, Zahlen und Unterstriche _ folgen. Umlaute sind nicht erlaubt.
  Matlab ist "case-sensitive", d.h. es wird zwischen Groß- und Kleinschreibung unterschieden (a≠A).

2.2.2 Grundlegende Befehle und Verknüpfungen

- Schreiben Sie grundsätzlich alle Matlabbefehle klein (z.B. if statt IF) auch wenn in der Matlab-Hilfe suggeriert wird, daß Befehle groß geschrieben werden können. In manchen Fällen funktioniert ein groß geschriebener Befehl, in manchen Fällen nicht.

- Arithmetische Ausdrücke können mit folgenden Zeichen gebildet werden:

+	: Addition;	**-**	: Subtraktion;
*****	: Multiplikation;	**/**	: Division;

^ : Potenzierung;

Es gelten die allgemeinen Verknüpfungsregeln (Punkt vor Strich), Klammern werden wie in allen Hochsprachen verwendet. Dies gilt uneingeschränkt allerdings nur für Skalare. Da Matlab auch die Zuweisung von Zahlenfeldern erlaubt, wird die allgemeine Nutzung von arithmetischen Ausdrücken später noch eingehender erläutert.

- Matlab besitzt einige vordefinierte Konstanten, z.B.

pi :	Π
i bzw. **j** :	imaginäre Einheit $i = j = \sqrt{-1}$.
inf:	unendlich, z.B. Ergebis der Rechenoperation 1 / 0.
nan:	Not a Number, undefinierter Ausdruck, z.B. Ergebnis der Operation 0 / 0.

- Einige grundlegende Matlab-Befehle:

clear: Löscht alle Variablen im Haupt-Speicher. Befehlsergänzungen (z.B. **clear function**) haben ein andere Bedeutung. Nähere Einzelheiten siehe "**help clear**".

home: Löscht das Matalb-Command-Window und setzt der Cursor in die linke obere Ecke des MCW.

% : Alle Zeichen hinter dem **%** - Zeichen werden bei der Programm-ausführung als Kommentar gewertet.

whos Mit diesem Befehl können alle im Matlabspeicher vorhandenen Variablen mit ihren Namen ihren Felddimensionen angezeigt werden. Nach Eingabe eines Variablennamens (und CR) wird der Inhalt der Variablen auf dem MCW ausgegeben.
Das gleiche, allerdings mit einer anschaulicheren Darstellungsstruktur in einem separaten Fenster, kann durch eine Klick auf den ***Pusch-Button 7*** (***Workspace Browser***) erreicht werden. Durch Anklicken des Variablennamens öffnet sich ein Editor und der Inhalt der Variablen kann betrachtet und auch geändert werden. (siehe Kapitel 2.7.2)

2.2.3 Matrizen und Zahlenfelder

<u>Matlab betrachtet alle Variablen grundsätzlich als Zahlenfelder</u>. Zahlenfelder können

- rechteckig: $\begin{bmatrix} 1 & 2 & 3 \\ 7.5 & 6 & 11 \\ 2 & 1 & 9 \end{bmatrix}$

- zeilenorientiert: $\begin{bmatrix}1 & 2 & 3 & 6.7\end{bmatrix}$

- spaltenorientiert $\begin{bmatrix}2\\4\\5.5\\9\end{bmatrix}$ sein, oder aus nur

- einer Zahl bestehen: [5.6] = 5.6 .

(Es existieren noch andere Formen, die am Ende dieses Kapitels kurz erwähnt werden).

Die Zahlen in den Feldern bezeichnet man als Feldelemente. Zahlenfelder müssen in Matlab nicht explizit dimensioniert werden, sie erhalten automatisch das bei der Eingabe gewählte Format.

2.2.3.1 Eingabe von Zahlenfeldern

Zahlenfelder werden mit rechteckigen Klammern eingegeben. Feldelemente einer Zeile werden durch ein Komma oder Leerzeichen, Feldelemente einer Spalte durch ein Semikolon oder einen (CR) voneinander getrennt. Im folgenden sind einige Beispiele aufgeführt:

- Eingabe von rechteckigen Zahlenfeldern: Die Eingabezeilen (z.B. in das MCW)

  a=[1**,**2.2**,**3**;**4**,**5**,**6.6**;**7**,**8**,**9] oder a=[1,2.2,3 (CR)
4,5,6.6 (CR)
7,8,9]

  erzeugen das Zahlenfeld

  $$a = \begin{bmatrix}1 & 2.2 & 3\\4 & 5 & 6.6\\7 & 8 & 9\end{bmatrix} \quad .$$

- Eingabe von zeilenorientiereten Zahlenfeldern: Die Eingabezeilen

  b=[1,2,3,4] oder b=[1 2 3 4] erzeugen b=[1 2 3 4].

- Eingabe von spaltenorientierten Zahlenfeldern: Die Eingabezeilen

c=[1;2;3;4] oder [1 (CR)
2 (CR)
3 (CR) erzeugen $c = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$
4]

.

- Eingabe von einfachen Zahlen (Skalaren): Die Eingaben

  d=[1] oder d=1 erzeugen d=1.

2.2.3.2 Zugriff auf und Veränderung von Elementen eines Zahlenfeldes

Auf Feldelemente eines Zahlenfeldes wird mittels Indizierung zugegriffen: Ist z.B. a ein mxn-Zahlenfeld, das heißt a hat m Zeilen und n Spalten, weist der Befehl

$$x = a\left(\overbrace{i}^{\substack{\text{Zeilen-}\\\text{index}}} \;,\; \underbrace{k}_{\substack{\text{Spalten-}\\\text{index}}} \right)$$

der Variablen x das Element in der i-ten Zeile und k-ten Spalte von a zu. Die <u>Indizes beginnen immer bei Eins</u> und nicht bei Null:

- Zugriff auf Feldelemente (z.B. eines existierenden 4x5-Zahlenfeldes a)

  x = a(3,4) weist x das Feldelement a(3,4) zu.
  x = a(1:4,3) weist x die ganze dritte Spalte von a zu. Da eine ganze Spalte adressiert wird, kann der Befehls auch mit x = a(:,3) verkürzt werden.
  x = a(2,1:5) weist x die ganze zweite Zeile von a zu. Auch hier kann, da die ganze Zeile zugewiesen wird, die Abkürzung x = a(2,:) benutzt werden.
  x = a(1:2,1:3)weist x das linke obere 2x3Teilfeld von a zu.

- Veränderung von Feldelementen eines existierenden Zahlenfeldes (z.B.: einer 3x3-Feldes a)

  a(2,3) = 7 weist dem Feldelement a(2,3) den Wert 7 zu und verändert sonst nichts.

a(1,1:3) = **ones** (1,3)	weist den Elementen der ersten Zeile von a Einsen zu. Auch hier kann, da eine ganze Zeile geändert wird, "1:3" durch ":" ersetzt werden.
a(2:3,2:3) = [5,6;7,8]	weist dem rechten oberen 2x2-Teilfeld von a die angegebenen Elemente zu.

Mit Hilfe der folgenden Befehle lassen sich Zahlenlfelder zu einem neuen Zahlenfeld kombinieren

- Aus den zeilenorientierten Zahlenfeldern a = [1,2,3,4] und b = [6,7,8,9] erzeugt der Befehl

  c = [a**,**b] das Zahlenfeld c = [1,2,3,4,6,7,8,9]; und der Befehl
  c = [a**;**b] das Zahlenfeld c = [1,2,3,4
6,7,8,9].

- Aus den spaltenorientierten Zahlenfeldern

  a = [1
2
3
4] und b = [6
7
8
9]

  erzeugt der Befehl

  c = [a**,**b] das Zahlenfeld c = [1 6
2 7
3 8
4 9]

  und der Befehl

  d = [a**;**b] das spaltenorientierte Zahlenfeld d = [1
2
3
4
6
7
8
9].

2.2.3.3 Hilfsbefehle zur Zahlenfeldgenerierung

Mit Hilfe der folgenden Befehle lassen sich Dimensionen von Zahlenfeldern feststellen:

[m,n] = **size**(a)	gibt die Zeilenanzahl m und die Spaltenanzahl n des Zahlenfeldes a an.

n=**length**(b)	gibt die Anzahl n der Elemente des zeilen- oder spaltenorientierten Zahlenfeldes b an.
b(n) = [ ];	tilgt das n. Element des zeilen- oder spaltenorientierten Zahlenfeldes b. (Redimensioniert (verkürzt) das Zahlenfeld b um ein Element).
y= **flipud**(x)	die Inhalte des spaltenorientierten Zahlenfeldes x erscheinen in umgekehrter Reihenfolge im spalten-orientierten Zahlenfeld y. (Mnemonik: u=up; d=down;)
y=**fliplr**(x)	die Inhalte des zeilenorientierten Zahlenfeldes x erscheinen in umgekehrter Reihenfolge im zeilen-orientierten Zahlenfeld y. (Mnemonik: l=left; r=right)

2.2.3.4 Automatische Zahlenfeldgenerierung

Die folgenden Befehle erzeugen automatisch bestimmte Zahlenfeldformen mit einem bestimmten (häufig gebrauchten) Inhalt:

x = **eye**(n)	erzeugt ein (quadratisches) nxn Zahelenfeld x, das bis auf die Hauptdiagonale mit Nullen besetzt ist. Die Hauptdiagonale (von links oben nach rechts unten) enthält Einsen ("Einheitsmatrx").
x = a **:** b **:** c	erzeugt ein zeilenorientiertes Zahlenfeld x x=[a, a+b, a+2b, a+3b, ... , c], wobei a,b und c Skalare sind.
x = **ones**(m,n)	erzeugt ein mxn-Zahlenfeld x mit allen Elementen gleich 1.
x = **zeros**(m,n)	erzeugt eine mxn-Zahlenfeld x mit allen Elementen gleich 0.
x = **ones** (**size** (b))	erzeugt ein Zahlenfeld von Einsen mit den Dimensionen des Zahlenfeldes b.
x = **logspace**(a,b,n)	erzeugt ein $\log_{10}$ gestaffeltes, n-Elemente langes, bei 10^a beginnenden und bei 10^b endendes zeilenorientiertes Zahlenfeld.

2.2.3.5 Die arithmetische Verknüpfung von Zahlenfeldern

Wie schon weiter vorn schon angemerkt wurde, besteht der große Unterschied zwischen Matlab und einer anderen "normalen" Hochsprachen darin, daß Matlab Variable ganz allgemein als Zahlenfelder auffaßt und auch diese arithmetisch verknüpfen kann.
Bei dieser Verknüpfung kommt es darauf an, als was man die zu verknüpfenden Zahlenfelder auffaßt:

- Betrachtet man die Zahlenfelder als Matrizen muß Ihre Verknüpfung nach den Regeln der Matrizenrechnung (siehe /1/ Anhang A2 "Grundlagen der Matrizenrechnung und linearer Gleichungssysteme") durchgeführt werden.

- Betrachtet man Zahlenfelder als solche und nennt sie z.B.: Arrays, wurde von den Autoren von Matlab eine weitere sehr sinnvolle Verknüpfung definiert, die sogenannte Arrayverknüpfung. Darunter wird die *komponentenweise Verknüpfung miteinander korrespondierender Arrayelemente* zweier Arrays verstanden.

Zum Beispiel führt die komponentenweise multiplikative Verknüpfung der beiden Arrays

$$a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}; \qquad b = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

auf das Lösungsarray

$$\begin{bmatrix} 1\cdot 5 & 2\cdot 6 \\ 3\cdot 7 & 4\cdot 8 \end{bmatrix} = \begin{bmatrix} 5 & 12 \\ 21 & 32 \end{bmatrix}.$$

Faßt man dagegen die beiden Zahlenfelder a und b als Matrizen auf und verknüpft sie nach den Gesetzen der Matrizenrechnung multiplikativ, würde man folgendes Produkt erhalten:

$$\begin{bmatrix} 1\cdot 5 + 2\cdot 7 & 1\cdot 6 + 2\cdot 8 \\ 3\cdot 5 + 4\cdot 7 & 3\cdot 6 + 4\cdot 8 \end{bmatrix} = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}.$$

Da eingegebenen Zahlenfelder nicht anzusehen ist, ob sie als Array oder als Matrix aufgefaßt werden sollen, d.h. ob sie komponentenweise oder nach den Gesetzen der Matrizenrechnung verknüpft werden sollen, besitzt Matlab ein Unterscheidungsmerkmal in den Verknüpfungsoperatoren:

*Faßt man Zahlenfelder als Arrays auf, d.h. man will sie **komponentenweise** mit einem der folgenden Operatoren*

*****, **/**, und **^**

*verknüpfen, muß man vor den Operator einen Punkt "**.**" setzen :*

.*, **./**, und **.^** .

*Bei der Addition und der Subtraktion **sollte** man den Punkt weglassen, da diese Operationen bei Matrizen auch komponentenweise definiert sind und Matlab sich nur dann eindeutig verhält.*

Bei Arrayverknüpfungen muß man darauf achten, daß die zu verknüpfenden Arrays vom gleichen Type sind, d.h. gleiche Anzahl von Zeilen und Spalten haben.

- *Faßt man <u>Zahlenfelder als Matrizen</u> auf und will sie nach deren Gesetzen der Matrizenrechnung mit den Operatoren*

  +, -, *, und **^**

  miteinander verknüpfen, benutzt man die angegebenen Operatoren ohne Zusatz und Matlab erkennt eine Matrixoperation.

  Zusatzbemerkungen zu den Matrixoperationen

  - Matlab kann die gewünschte Verknüpfung nur vornehmen, wenn sie nach den Gesetzen der Matrizenrechnung überhaupt möglich ist,
  - bei der Potenzierung einer Matrix muß der Exponent ein Skalar sein,
  - in der Matrizenrechnung ist die Division nicht definiert. An ihrer Stelle wird mit der inversen Matrix gearbeitet. Matlab stellt zwar auch einen (sehr schnellen) "Divisionsoperator" für Matrizen zur Verfügung, aus didaktischen Gründen wollen wir aber auf seine Anwendung verzichten.

- **Beispiel zur Verknüpfung von Matrizen.**
  Bei der Lösung linearer Gleichungssysteme der Form

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

muß die inverse Matrix zur Matrix A gebildet und nach den Gesetzen der Matrizenrechnung von links mit dem Vektor b multipliziert werden, um den Vektor x der Unbekannten zu erhalten:

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

Unter Matlab muß dazu folgende Befehlsfolge benutzt werden:

```
a = ... ;       % vorgegebene Matrix a
b = ... ;       % vorgegebener Vektor b
ai = inv(a);    % Invertierung von a
x = ai * b      % Matrix-Multiplikation
```

- **Beispiel zur Verknüpfung von Arrays.**
  Arrayverküpfungen können vorteilhaft zur Vermeidung von Schleifen bei der Berechnung z.B. von Funktionen eingesetzt werden. Mit Hilfe der Arrayverküpfung kann die folgende bei "normalen" Hochsprachen genutzte Programmsequenz zur Berechnung der Funktionswerte eines Polynoms y=f(x) im Bereich $0 \leq x \leq 10$

```
for x = 0 : 0.1 : 10
     index = 10 * x + 1;
     y (index) = 2 * x ^2 + 3 * x + 4;
end
```

unter Matlab durch folgende Kodefolge ohne Schleife ersetzt werden:

```
x = 0:0.1:10;
y = 2 * x .^2 + 3 * x + 4;  .
```

Die Bildung der Quadrate des Vektors x wird in einem Zug durch die Arrayverknüpfung x.^2 vorgenommen.
Die zweite Programmierungs-Möglichkeit sollte bei Matlab-Programmen immer genutzt werden, da der Kode übersichtlicher ist und wesentlich schneller läuft als eine Schleife.

Da sich im technisch / naturwissenschaftlichen Sprachgebrauch die Begriffe Matrix und Vektor besser etabliert haben, als der Begriff Zahlenfeld, wollen wir in folgenden ohne Beschränkung der Allgemeinheit folgende Begriffsbildungen benutzen:

Matrix	*statt*	*Zahlenfeld*
Zeilenvektor	*statt*	*zeilenorientiertes Zahlenfeld*
Spaltenvektor	*statt*	*spaltenorientiertes Zahlenfeld.*

Matlab arbeitet darüber hinaus auch mit Zahlenfeldern, deren Dimensionen größer als zwei sind, z.B. A(2,4,2). Wir benutzen in dieser Einführung diese Zahlenfelder nicht. Nähere Einzelheiten können unter ***helpdesk -> Using Matlab -> Multidimensioanl Arrays*** nachgelsen werden.

2.2.4 Structures und Cell-Arrays

Strukturen (structures, structs) und Cell-Arrays sind Datenbehälter (Speicher) für Daten verschiedenster Art. Sie dienen zur übersichtlichen Verwaltung dieser Daten bis hin zur Erstellung einfacher Datenbanken.

2.2.4.1 Strukturen

Strukturen werden mit dem Befehl **struct** und folgender Syntax erzeugt:

MeineStruktur = **struct** ('name1', wert1, 'name2' ,wert2 , ...);

Ein Zugriff auf die in einer Struktur gespeicherten Daten erfolgt mit einem Punkt-Operator (**.**), z.B.:

Feldinhalt1 = MeineStruktur**.**name1;

Damit wird der Variablen "Feldinhalt1" der Inhalt des Feldes "name1" der Struktur "MeineStruktur" nämlich "wert1" zugewiesen.

Mit dem Befehl **whos** oder mittels des "Workspace-Browser"-Buttons kann festgestellt werden, welche Variablen und welche Datentypen/Datenstrukturen sich im Matlab-Speicher befinden (vergleiche das Programm "demo_struct_cell.m" am Ende des Kapitels 2.2.4):

```
» whos
  Name                          Size          Bytes  Class

  Arztpraxis                    2x2            1610  cell array
  Datum_Pat2                    1x1             396  struct array
  Geraeteindex                  1x4              32  double array
  Letzte_Untersuchung           0x0               0  double array
  Patient                       1x2            1280  struct array
  Patientendatei                1x2            1280  struct array
  Patientenname1                1x12             24  char array
  Untersuchungen                2x2              32  double array
  Zahlbetrag                    1x1               8  double array

Grand total is 176 elements using 4662 bytes
```

Bei der Variablen "Patient" handelt es sich z.B. um eine Struktur (struct array). "Size" 1x2 ist die Array-Größe, sie deutet darauf hin, daß es sich um zwei Patienten, nämlich Patient(1) und Patient(2) handelt. "Bytes" gibt den Speicherverbrauch der entsprechenden Variablen im Matlab-Arbeitsspeicher an.

Durch Eingabe des Variablennamens "Patient (CR)" werden neben der Größeninformation der Struktur auch die inneren Feldnamen angegeben:

```
» Patient

Patient =

1x2 struct array with fields:
     Name
     Rechnungsbetrag
     Untersuchungen
     Untersuchungsdatum
```

Auf diese Felder kann mit dem Punktoperator zugegriffen werden. Das Programm "demo_struct_cell.m" demonstriert dies und zeigt weiterhin, wie man durch Zuweisungs-Anweisungen Strukturen erweitern kann.

2.2.4.2 Cell-Arrays

Cell-Arrays sind eine spezielle Klasse von Matlab-Arrays, deren Elemente als Zellen (cells) bezeichnet werden. Zellen können wiederum beliebige Datentypen und Datenstrukturen enthalten.
Cell-Arrays werden mit dem Befehl **cell** erzeugt:

MeinCellArray = **cell {**2, 2**};**

Dieses Cell-Array besitzt 2x2 = 4 (noch leere) Speicherzellen. Die Dimensionierung ist beliebig und kann über zwei Dimensionen hinausgehen. Die Zellen können durch entsprechende *Indizierung in geschweiften Klammern* mit Daten gefüllt werden, z.B.:

MeinCellArray **{**1,1**}** = 'Mit freundlichen Gruessen';
MeinCellArray **{**1,2**}** = [1, 2, 3, 4]; .

Ebenfalls durch Indizierung in geschweiften Klammern kann man auf die Inhalte der Zellen eines Cell-Arrays zugreifen, z.B.:

Inhalt1 = MeinCellArray **{**1,1**}** gibt aus
Inhalt1 =
Mit freundlichen Gruessen

In dem folgenden Programm "demo_struct_cell.m" wird die Speicherung weiter Datenstrukturen in Cell-Arrays demonstriert.
Durch Entfernen der Kommentarzeichen vor **echo on** und **echo off** gibt das laufende Programm jeweils nacheinander dem Kommentar, die Kodezeile und das Berechnungsergebnis aus. Damit kann das Ursache-/Wirkungsverhalten des Programms sehr schön demonstriert werden.

```
%  File “Demo sruct_cell“
%
% Programm zur Demonstration der Funktionsweise von
%  - Strukturen (strutures) und
%  - Cell Arrays

% Leeren des Matlab-Speichers, MCW säubern und schließen
% aller geöffneten Grafik-Fenster
clear, home, close all
```

```
%echo on
% 1. STRUKTUREN
% Erzeugung der Struktur "Patient" mit dem Befehl "struct".
Patient=struct('Name','Paul Schmidt','Rechnungsbetrag'...
   ,123.20,'Untersuchungen',[1,4;15,8]);

% Erweiterung der Struktur Patient um einen weiteren
% Patienten.
% (Patient ist dann identisch mit Patient(1)).
Patient(2)=struct('Name','Franz Gans','Rechnungsbetrag'...
   ,1200.00,'Untersuchungen',[4,10]);


% Inhalte (Feldnamen)der Struktur Patient
Patient

% Beispielzugriff auf die Feldelemente des Patienten(1)zur
% Anzeige des Inhalts
Patientenname1=Patient(1).Name
Zahlbetrag=Patient(1).Rechnungsbetrag
Untersuchungen=Patient(1).Untersuchungen

% Erzeugung einer Struktur "Datum"
Datum_Pat2=struct('Tag',10,'Monat',09,'Jahr',98);

% Einfügung der Struktur "Datum_Pat2" als Unterstruktur
% (Strukturschachtelung)in die Struktur von "Patient(2)".
% (Möglichkeit 2 zur Erzeugung von Strukturen
% durch Zuweisungsanweisungen).
Patient(2).Untersuchungsdatum=Datum_Pat2;

% Anzeige der erweiterten Struktur "Patient"
Patient

% Zugriff auf die Unterstruktur "Untersuchungdatum" von
% Patient(2).
Letzte_Untersuchung=Patient(2).Untersuchungsdatum

% Zugriff auf die Unterstruktur "Untersuchungdatum" von
% Patient(1),die leer sein muß, weil ihr kein Wert
% zugewiesen wurde.
Letzte_Untersuchung=Patient(1).Untersuchungsdatum


% 2. CELL-ARRAYS
% Erzeugung eines Cell-Arrays "Arztpraxis" mit 4 Zellen
% mit dem Befehl "cell".
Arztpraxis=cell(2,2);

% Besetzung der Zellen
Arztpraxis{1,1}='Dr. Pille';       % Praxisbesitzer.
Arztpraxis{2,1}=[1,3,2,5];         % Geraeteindex.
Arztpraxis{2,2}=Patient;           % die oben erzeugte
                                   % Patientenstruktur.
% Die Zelle Arztpraxis{1,2} wurde leer gelassen.
```

```
% Anzeige des Inhalts des Cell-Arrays "Arzrpraxis"
Arztpraxis

% Zugriff auf die nicht lesbaren Inhalte
Geraeteindex=Arztpraxis{2,1}
Patientendatei=Arztpraxis{2,2}
%echo off
```

2.3 Mathematische Funktionen

2.3.1 Grundlegende mathematische Funktionen

Matlab besitzt, wie alle Hochsprachen, u.a. folgende mathematischen Funktionen, die jedoch auch Matrizen als Argumente besitzen dürfen (d.h. das folgende Argument x kann eine Matrix sein) :

exp(x) :	e^x ; e-Funktion
log(x):	ln x; natürlicher Logarithmus
log10(x) :	log x; dekadischer Logarithmus
sqrt(x) :	$\sqrt{x}$; Quadratwurzel
abs(x) :	$\lvert x \rvert$; Betrag von x
sin(x); **cos**(x); **tan**(x) :	Trigonometrische Funktionen, das Argument x muß in rad eingegeben werden
asin(x); **acos**(x); **atan**(x) :	Umkehrfunktionen der trigonometrischen Funktionen
sinh(x); **cosh**(x); **tanh**(x) :	Hyperbolische Funktionen
fix(x) :	Rundung von x zum nächsten Integer Richtung Null
round(x):	kaufmännische Rundung von x

2.3.2 Höhere mathematische Funktionen

a = **diff** (x) : Wenn $x=[x_1,x_2,x_3,x_4,x_5,...,x_n]$ ist, dann ist $a=[x_2-x_1, x_3-x_2, x_4-x_3, ..., x_n-x_{n-1}]$. Dieser Befehl kann z.B. zur näherungsweisen Differenzierung von Funktionen benutzt werden:

```
T=0.1;
t=0:T:2*pi;
y=sin (t);
ys=diff(y)/T;  % genäherte Differentiation
plot(y),hold on
plot(ys)
hold off
```

a = **sum**(x) : a ist die Summe der Elemente des Vektors x. Dieser Befehl kann z.B. zur näherungsweisen Berechnung eines bestimmten Integrals benutzt werden:

```
T=0.1;
t=0:T:pi;
y=sin(t);
integral=sum(y)*T  % genähert. best. Integral
```

a=**cumsum**(b): a ist der Vektor der kumulierten Summe der Werte der Elemente von b. Mittels "cumsum" können Funktionen im Sinne eines unbestimmten Integrals integriert werden:

```
T=0.1;
t=0:T:2*pi;
u=cos(t);
int_u=cumsum(u)*T; % genähertes unbest. Integral
plot(u);
hold on;
plot(int_u);
hold off;
```

2.3.3 Weitere Matrixoperationen

a = **inv**(b) :	Matrixinversion, a ist die inverse Matrix von b.
a = **det**(b) :	a ist der Wert der Determinante der Matrix b.
a = b**'** :	a ist die Transponierte der Matrix b.
a = **max**(b) :	a ist das größte Element des Vektors b (vergleiche "help Max").
a = **min**(b) :	a ist das kleinste Element des Vektors b (vergleiche "help min").
a = **mean**(b) :	a ist das arithmetische Mittel der Elemente des Vektors b.
a = **sort**(b) :	a ist der vom kleinsten bis zum größten Element sortierte Vektor b.

2.3.4 Komplexe Zahlen

Komplexe Zahlen werden i.a. in Komponentenform eingegeben:

z = a + j*b; z.B. z = 3 + j*4.

Mit den folgenden Befehlen können komplexe Zahlen manipuliert bzw. in andere Schreibweisen, z.B. die Betrags- und Phasenform, umgeformt werden::

real(z) : Berechnet der Realteil einer komplexen Zahl z
imag(z) : Berechnet den Imaginärteil einer komplexen Zahl z
conj(z) : Berechnet die konjugiert komplexe Zahl zu z

abs(z) : Berechnet den Betrag einer komplexen Zahl:

$$r = |z| = \sqrt{a^2 + b^2}$$

angle(z) :Berechnet den Phasenwinkel einer komplexen Zahl (in rad):

$$\text{angle}(z) = \arctan\left(\frac{b}{a}\right)(\pm \Pi \text{ , falls } a < 0)$$

2.3.5 Polynome

Ein Polynom

$$a_n s^n + a_{n-1} s^{n-1} + \cdots + a_2 s^2 + a_1 s^1 + a_0 ; \qquad a_i = \text{konst.}$$

wird in Matlab durch den Vektor der Koeffizienten a_i, geordnet nach fallenden Potenzen von s, eingegeben:

$$a = [\ a_n, a_{n-1}, \cdots, a_2, a_1, a_0\ ] ;$$

Beispiel: $4s^3 + 2s^2 + 3 \;\Rightarrow\; \text{Matlab-Eingabe} \quad a = [4,2,0,3].$

Mit dieser Vereinbarung können folgende Operationen mit Polynomen vorgenommen werden:

c = **conv**(a,b) :	c ist der Koeffizientenvektor des Produktes der Polynome a und b.
[q,r] = **deconv**(b,a):	Polynomdivision: $\underline{b} : \underline{a} = \underline{q} + \frac{\underline{r}}{\underline{a}}$
n =**roots**(a) :	Nullstellenberechnung von Polynomen, der Vektor n enthält sämtliche Nullstellen (Wurzeln = roots) des Polynoms a.
a = **poly**(n) :	Umkehrfunktion von "roots", berechnet aus den gegebenen Nullstellen (n) den dazugehörige Vektor a der Polynom-koeffizienten.
y=**polyval**(a,x) :	y ist der Vektor der Funktionswerte des Polynoms mit dem Koeffizientenvektor a an den Stellen des Vektors x.

2.4 Programmlauf-Kontrolle

Matlab verfügt über zwei Verzweigungsbehle, **if** und **switch**, für Fall-unterscheidungen, und zwei Schleifenbefehle, **for** und **while**, mit denen bestimmte Anweisungen mehrfach durchlaufen werdn können. Zur Bildung von dabei benutzten Ausdrücken stehen folgende Vergleichs- und logischen Operatoren zur Verfügung:

Vergleichsoperatoren

==	**eq(a,b)**	gleich
~=	**ne(a,b)**	ungleich
<	**lt(a,b)**	kleiner
<=	**le(a,b)**	kleiner gleich
>	**gt(a,b)**	größer
>=	**ge(a,b)**	größer gleich

Logische Operatoren

<table>
<tr><td>~</td><td>not(a)</td><td>Negation</td></tr>
<tr><td>&</td><td>and(a,b)</td><td>UND</td></tr>
<tr><td>|</td><td>or(a,b)</td><td>ODER</td></tr>
<tr><td></td><td>xor(a,b)</td><td>Exlusiv ODER</td></tr>
</table>

Dabei können alternativ die Zeichen (z.B. a **<=** b) oder die Befehle (**le**(a,b)) verwendet werden.

Jede Verzweigung oder Schleife muß mit einem **end** abgeschlossen werden.
Weiterhin stehen ein **break**- und ein **continue**-Befehl (näheres siehe help) zur Ablaufsteuerung zur Verfügung.

- Eine **for-Schleife** hat folgende allgemeine Struktur:

```
for Variable = Ausdruck,
    Befehl;
    Befehl;
    ... ;
end
```

Beispiel:

```
N=5;
for I = 1:N,
   for J = 1:N,
      A(I,J) = 1/(I+J-1);
   end
end
```

- Eine **while-Schleife** hat folgende allgemeine Struktur:

```
while Ausdruck,
      Befehl;
      Befehl;
      ... ;
end
```

Beispiel:

```
N=0;
while N <= 10,
     disp(N);
     N = N+1;
end
```

- Eine **if-Verzeigung** hat folgende Struktur:

```
if Ausdruck,
   Befehle;
elseif Ausdruck,
   Befehle;
else
   Befehle;
end
```

Beispiel:

```
if I == J
   A(I,J) = 2;
elseif abs(I-J) == 1
   A(I,J) = -1;
else
   A(I,J) = 0;
end
```

- Eine **switch-Verzweigung** hat folgende Struktur:

```
switch Ausdruck,
    case Ausdruck,
        Befehle;
    case Ausdruck,
        Befehle;
    ...

    otherwise
        Befehle
end
```

Beispiel:

```
a=2;
switch a
  case 1
    disp('a=1')
  case 2
    disp('a=2')
  otherwise
    disp('unbekannt')
end
```

2.5 m-Files: Skripts und Functions

Wie wir schon weiter vorn erwähnt haben, besitzt Matlab auf MCW-Ebene einen Zeileneditor, mit dem zeilenweise Kommandos eingegeben werden können. Nach

Betätigung der Return-Taste (CR) führt Matlab den Befehl sofort aus. Matlab ist aber auch in der Lage, Programme abzuarbeiten, die in einem File (Datei) gespeichert sind. Files, die Matlab-Befehle enthalten, werden m-Files genannt, da ihre Filenamen die Extension ***.m*** tragen, z.B. bessel.m könnte eine Matlab-Funktion beinhalten, die Besselfunktionen berechnet.

Ein m-File besteht aus einer Sequenz normaler Matlab-Befehle, inklusive anderer m-File Aufrufe. (Der größte Teil der Matlab-Befehle sind solche m-Files, nur der Grundvorrat der Matlab-Befehle sind sogenannte "built in functions".)

M-Files lassen sich in zwei Gruppen einteilen. Die sogenannten *Script-Files* werden dazu benutzt, lange Sequenzen von Befehlen einzugeben. Die andere Gruppe sind die *Function-Files*, sie erlauben es z.B., neben den existierenden Matlab-Befehlen neue, eigene Matlab-Funktionen zu entwickeln. Beide Gruppen von m-Files sind normale ASCII-Textfiles, die mit dem Matlab-Editor geschrieben oder geändert werden können.

- Skript-Files sind häufig Hauptprogramme mit Benutzer-Kommunikationsschnittstelle (Dateneingabe, Ergebnisausgabe). Variable in Skript-Files sind global definiert.
- Function-Files sind solche m-Files, in deren erster Zeile das Wort "function" auftritt und es so als Function-File definiert. Ein Function-File kommuniziert mit dem aufrufenden Programm über Parameterlisten. Alle Variablen innerhalb eines Function-Files sind nur lokal definiert.

  Das folgende Beispiel eines Function-Files berechnet von einer Zahlenfolge, die im Vektor x steht, den Mittelwert "mittel" und die Standardabweichung "stdabw" :

```
function[mittel,stdabw] = stat(x)
n = max(size(x));
mittel = sum(x)/n;
m_vek = ones(size(n))*mittel;
stdabw = sqrt((1/(n-1))*sum((x-m_vek).^2));
```

  Das m-File muß den gleichen Namen tragen, wie die function selbst (in unserem Beispiel "stat.m"). Es kann dann in folgender Sequenz aufgerufen werden:

⋮

Bereitstellung einer Zahlenfolge
innerhalb eines Vektors x

⋮

[m,sa] = stat (x) %Aufruf des function-Files

⋮

⋮

Nutzung von m (Mittelwert von x)
und sa (Standardabweichung von x)
wie benötigt.

⋮

Die Function-Deklaration hat folgenden formalen Aufbau:

function [output-Parameterliste] = Funktionsname (Input-Parameterliste)

wobei der Funktionsname frei wählbar ist. Funktionsname und m-File-Name müssen identisch sein. Sowohl die output-Parameterliste als auch die input-Parameterliste können Matrizen, Vektoren und Skalare (auch gemischt) enthalten.
<u>Die Parameterübergabe erfolgt "by value"</u>.

Functions werden beim ersten Gebrauch vorkompiliert und im Matlab-RAM-Speicher abgelegt. Bei allen folgenden Aufrufen dieser Functions wird zur Erhöhung der Programm-Ablaufgeschwindigkeit auf diese vorkompilierte Form zurückgegriffen.
Mit dem Befehl **pcode Funktionsname** werden Functions vorkompiliert als sogenanntes "p-file", d.h. mit der Namensergänzung .p, im aktuellen Arbeitsverzeichnis des Programmspeichers (i.a. die Festplatte) abgelegt. Beim ersten Programmlauf wird dann sofort auf diese vorkompilierte Version zurückgegriffen. Entfernt man die m-File Versionen der Functions aus dem Matlab-Verzeichnis (nicht nur aus dem Arbeitsverzeichnis) kann der Kode der Functions nicht mehr eingesehen werden. Damit kann man bei Weitergabe eines Programms sein geistiges Eigentum schützen (nähere Einzelheiten siehe **help pcode** und **help clear**).

Der Leser diskutiere zum Abschluß, welche Ergebnisse sich in den Variablen k,l,m und n befinden, wenn das folgende Function-File tstfun.m von dem darüberstehenden Skriptfile aufgerufen wird:

```
a=1;
b=2;
c=[3,4,5];
d=[6;7;8;9];
[k,l,m,n]=tstfun(a,b,c,d);
```

```
function[out1,out2,out3,out4]=tstfun(in1,in2,in3,in4)
out1=in2;
out2=in1;
out3=in4;
out4=in3;
```

2.6 Benutzerkommunikation

2.6.1 Alphanumerische Ein- und Ausgaben

Die alphanumerische Benutzerkommunikation kann nach den Ansprüchen des Nutzers sehr einfach bis sehr komfortabel realisiert werden.

- **Alphanumerische Eingaben**

Eine numerische Eingabe-Abfrage (also die Abfrage nach einer Zahl) erfolgt mit folgendem Befehl

x = **input ('**Text**');**.

Dieser Befehl gibt den String "Text" aus und erwartet eine Eingabe, die der Variablen x zugewiesen wird. Bei Matrix- und Vektoreingaben müssen die üblichen Klammern [ ] mit eingegeben werden.

Ein String-Input (also Buchstaben) kann durch folgende Parametrierung abgefragt werden:

Antwort=**input ('**Frage**' , 's');**

Die Textausgabe kann dabei mit dem Steuerzeichen **\n** (Zeilenumbruch) auf auf mehrere Zeilen verlängert werden. (Die eckige Klammer muß gesetzt werden, weil ein Vektor von zwei Strings ausgegeben werden soll):

x = **input (['**Text1 **\n ' , '**Text2**'])**;

Auch vorher berechnete numerische Werte (hier der Wert der Variablen „a)“ können in den Ausgabetext aufgenommen werden:

input([' Text1**',num2str(**a**),'** Text2**'])**;

Der Befehl **num2str** wandelt den numerischen Wert von a, in einen Zahlenstring, um dann diesen auszugeben.

- **Alphanumerische Ausgaben**

Die einfachste Form der alphanumerischen Ausgabe ist das Weglassen des Semikolons hinter einer Anweisungszuweisung:

$$\vdots$$
$$\left.\begin{array}{r} x = 423.6\,; \\ \text{Geschwindigkeit} = x \end{array}\right\} \begin{array}{c}\text{führt zur} \\ \text{Ausgabe}\end{array} \left\{\begin{array}{l} \text{Geschwindigkeit} = \\ \quad 423.6000 \end{array}\right.$$
$$\vdots$$

Der Befehl **disp** erlaubt eine Ausgabe mit den folgenden Parametrierungen (Vergleiche den Befehl **input**):

disp('Text**')** :
Gibt den String "Text" aus.

disp([' Text1**',num2str(**a**),'** Text2**']);**
Gibt die Strings 'Text1' und 'Text2' und dazwischen den numerischen Wert der Variablen a aus. a kann ein Skalar, Vektor oder eine Matrix sein.

Auch die im nächsten Kapitel zu besprechenden Beschriftungsbefehle für Graphiken, **xlabel**, **ylabel** und **text**, können auch wie die vorangehenden Befehle parametriert werden.
Mit dem Befehl **home** kann das Matlab Command Window gelöscht und der Cursor in die linke, obere Ecke des Bildschirms plaziert werden..

2.6.2 Graphische Ausgaben

Im Gegensatz zu den vorangehend genannten numerischen Ausgaben sind die grafischen Ausgaben von Matlab sehr komfortabel. In dieser Übersicht kann wieder nur ein kleiner Teil dieser Leistungen beschrieben werden.

plot (y) : Stellt die in y gespeicherte Wertefolge als Polygonzug über der Indizierung von y dar, d.h. wenn y aus 85 Elementen besteht, wird die Abszisse von 1 bis 85 beschriftet. Die Skalierung der Ordinate führt Matlab auch selbständig aus. Mit Hilfe des Befehls "axis" (ausgeführt nach dem Plot-Befehl) können sowohl die Abszisse als auch die Ordinate vom Benutzer skaliert werden.

plot (x,y) : Stellt die Wertefolge y über der Wertefolge x dar, x wird i.A. eine Zeitachse sein, die mit dem Befehl "x=a:b:c" erzeugt wurde.

plot (x1,y1,x2,y2,...): Stellt die Wertefolgen y1 über x1, y2 über x2, usw. in einer Grafik dar. Die einzelnen Kurven werden automatisch farbig unterschieden. (Für eigene Wahl der Farbgebung und Strichart siehe "help plot")

axis (v) : Skaliert eine vorangegangene Plot-Ausgabe nach Maßgabe des Vektors v=[x_{min}, x_{max}, y_{min}, y_{max}].

semilogx (x,y): Ist ein Plot-Befehl, der die Wertefolge y über einer dekadisch logarithmischen geteilten Abszisse darstellt. Der Vektor x muß vorher mit dem Befehl "logspace" erzeugt werden.

grid on : Legt ein Gitternetz über das geplottete Koordinatensystem. Muß nach einem Plot Befehl ausgeführt werden.

title('Text**')** : Gibt den String "Text" als Grafiküberschrift aus. Muß nach einem Plot Befehl ausgeführt werden.

xlabel('Text**') ,**
ylabel('Text**')** : Gibt den String 'Text' als Beschriftung der x -Achse (Abszisse) bzw. der y-Achse (Ordinate) aus. Muß nach einem Plot Befehl ausgeführt werden.

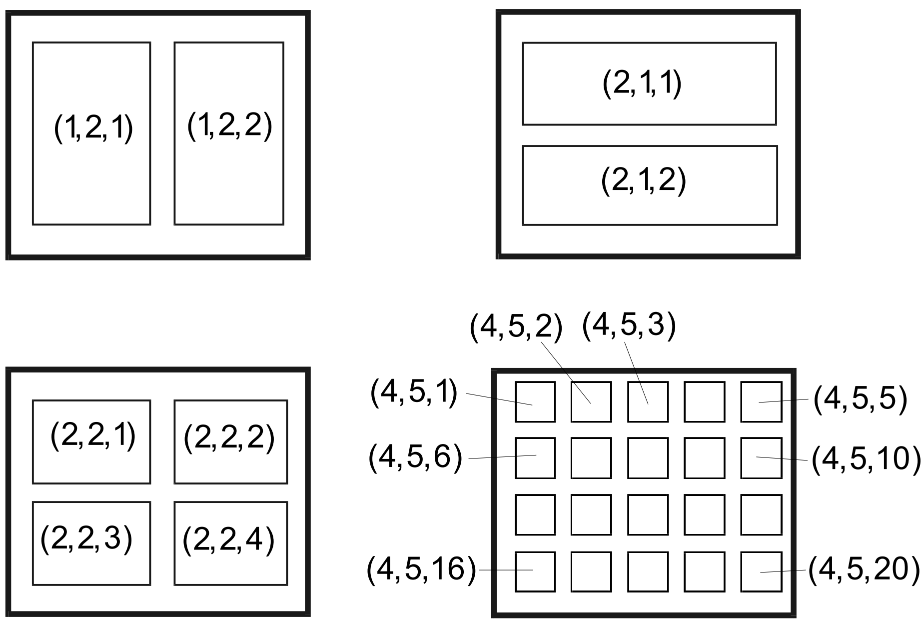
text : Beschriftet Plots an beliebigen Stellen. Nähere Einzelheiten siehe "help text". Muß nach einem Plot Befehl ausgeführt werden.

hold on: Hält den aktuellen Bildschirm und ermöglicht so aufeinander-folgende Ausgaben von Plots in ein Koordinatensystem.

hold off: Setzt "hold on" zurück.

pause : Stoppt den Programmlauf an der Stelle, wo der Befehl "pause" gefunden wird. Wird häufig nach Ausgabe eines Plots benutzt, um die Grafik betrachten zu können.

subplot(m,n,p) : Teilt den Grafikbildschirm in Teilbilder auf. Dabei werden die Teilbilder wie Matrix-Elemente adressiert. m gibt die Anzahl der Teilbilder in horizontaler Richtung (Zeilen) und n die Teilbilder in vertikaler Richtung (Spalten) an. Mit der Variablen p wird festgelegt, in welches Teilbild der nächste Plot ausgeführt werden soll.



Die Zählung von p beginnt beim linken oberen Teilplot und wird über die 1., 2. bis zum Ende der m. Zeile fortgesetzt.

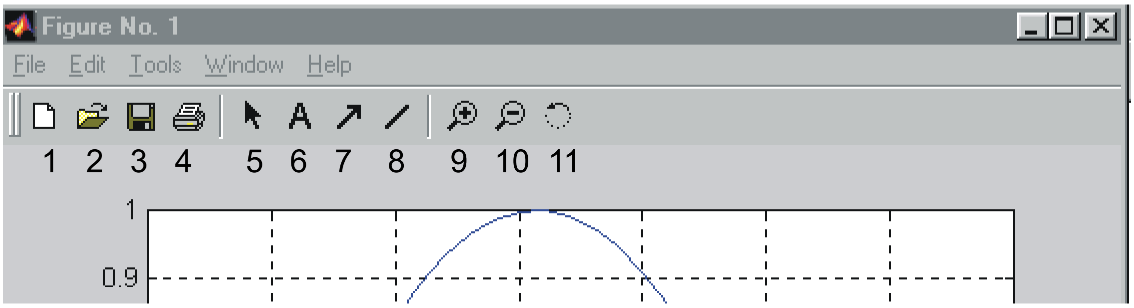
Die obigen Bilder zeigen beispielhaft, welche Bilschirmaufteilungen sich bei verschiedenen Parametern (m,n,p) ergeben.
Nach der Festlegung des Plotbereiches mit dem subplot-Befehl muß mit einem nachfolgenden Plot-Befehl der Fenster-Inhalt zugewiesen werden.

figure(n): Neben der Tatsache, daß auf einem Bildschirm , wie vorangehend darstellt wurde, mehrere Plots angeordnet werden können, besteht mit dem Befehl figure(n) darüber hinaus die Möglichkeit, beliebig viele Plot-Bildschirme zu erzeugen: n = 1, 2, 3, ... Zwischen diesen Bildschirmen kann später mit der Tastenkombination ALT+TAB umgeschaltet werden. (figure(1) erzeugt den ersten, figure(2) den zweiten Bildschirm, usw.)

close: Schließt das aktuelle (zuletzt benutzte) Grafikfenster. Kann mit anderen Befehlsergänzungen spezifischer eingesetzt werden. Nähere Einzelheiten siehe **help close**.

Auch für dreidimensionale Plots stehen eine Reihe von Befehlen zur Verfügung, die hier nicht weiter diskutiert werden sollen.

Auch im Grafikbildschirm selbst lassen sich nach Ausgabe eines Plots noch Manipulationen vornehmen (Vergleiche abgebildeten Ausschnitt des Grafikfensters):



Ausschnitt eines Grafik-Bildschirms

Im ***Pulldown-Menue Edit*** sind die Zeilen ***Copy Options*** und ***Copy Figure*** wichtig, wenn man eine Grafik in ein Textverarbeitungssystem kopieren will.
Es wird empfohlen bei ***Copy Options -> Windows Metafile*** und ***White Background*** und ***Match Figure Screen Size*** anzuklicken. ***Copy Figure*** legt dann den aktuellen Plot mit Achsen und Achsenbeschriftung in der Zwischenablage ab, von wo er z.B. in einTextverarbeitungssystem übernommen werden kann.

Durch Anklicken des ***Pfeiles*** **(5)** lassen sich der Plot (***Line***), die Achsen und Hilfslinien (***Axes***) und die Achsenbeschriftung und ggf. andere Beschriftungen (***Text***) markieren. Im **Pulldown-Menue *Tools*** lassen sich dann die entsprechenden Eigenschaften (***Properties***) verändern.
Mit dem ***Pushbutton 6*** läßt sich an beliebige Stellen der Grafik Text einfügen, mit **7** lassen sich Pfeile und mit **8** Linien in die Grafik einfügen. **9** vergrößert mit Klicks in die Grafik den Plot, **10** verkleinert ihn entsprechend. Mit **11** kann man 3-dimensionale Plots rotieren.

2.6.3 Ein grafisches Benutzerinterface (GUI)

Eine auch gehobenen Ansprüchen gerecht werdende Form der Benutzer-kommunikation (mit alphanumerischer Ein- und Ausgabe, Mausabfrage und Grafikausgabe bietet das ***Gaphical User Interface (GUI)*** .
Besondere Werkzeuge erlauben die Konstruktion einer in den Maßen frei wählbaren Ein- und Ausgabemaske mit verschiedensten Bedienelementen (z.B. Buttons, Radiobuttons, Pulldown-Menues). Bei Mausklicks und Keyboard-Eingaben werden ereignisgesteuert Programmsegmente ausgeführt, so daß eine interaktive Kommunikation zwischen Matlab-Programm und Nutzer möglich wird.
Nähere Einzelheiten zur GUI-Programmierung findet der Leser in /8/

2.6.4 Speichern und Laden von Daten

Zur Speicherung von Daten, die mit einem Matlab-Programm erzeugt wurden, kann der Befehl "**save**" benutzt werde. Die Anweisung

save filename x y z;

speichert die Variablen x, y und z (die wiederum Skalare, Vektoren oder Matrizen sein können) unter dem Namen "filename.mat" im aktuell eingestellten Verzeichnis. Der Befehl "**save**" kann noch mit anderen Parametern benutzt werden, nähere Einzelheiten können mit ***help save*** nachgelesen werden.
Zum Laden so gespeicherter Daten in das MCW bzw. in ein laufendes Matlab-Programm wird er Befehl "**load**" benutzt. Die Anweisung

load filename

lädt die Variablen aus der Datei "filename" mit ihren bei "**save**" benutzten Variablen-namen in den Matlab-Arbeitsspeicher.

Mit **save** und **load** können auch *ASCII-Files* gespeichert und geladen werden. Nähere Einzelheiten siehe **help save**, bzw. **load**.

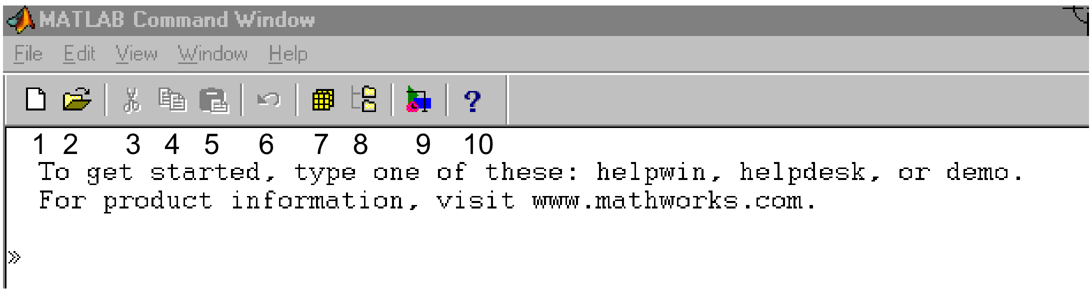
Matlab kennt noch weitere C-ähnliche Lade und Speicherbefehle, wie z.B.

fopen, **fread**, **fwrite** und **fclose.**

Für nähere Einzelheiten siehe "**help ...**".

2.7 Einzelheiten der Matlab-Arbeitsumgebung

In Kapitel 2.1 wurde das Matlab-Command-Window (MCW) eingeführt:



Ein Ausschnitt aus dem Matlab-Command-Window

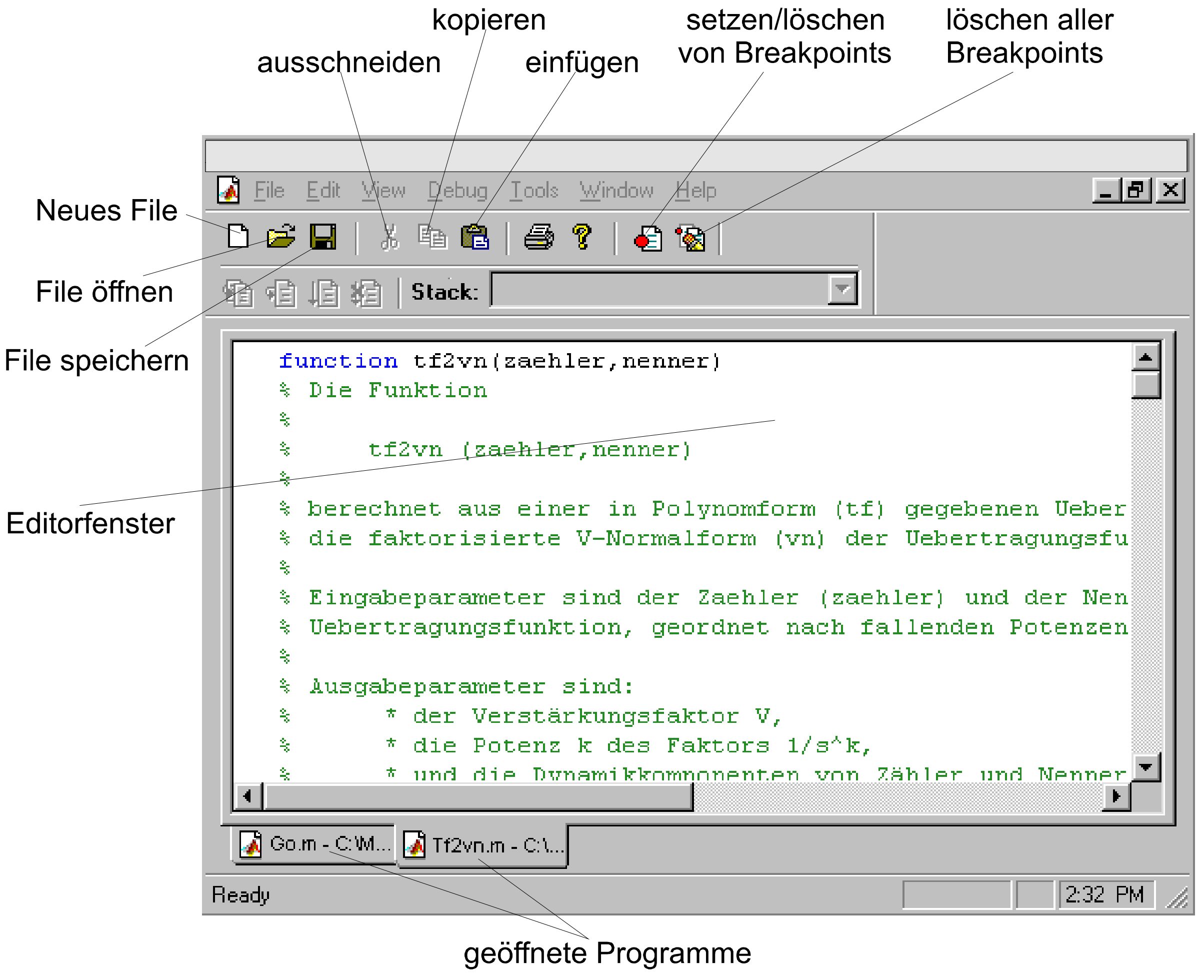
An dieser Stelle sollen die Funktionen

Matlab-Editor/-Debugger	Button 1
Workspace Browswer	Button 7
Path Browswer	Button 8

die mit den aufgeführten Buttons im MCW aufrufbar sind näher besprochen werden. Auf die Simulink-Library (Button 9) gehen wir ausführlich in Kapitel 4 ein.

2.7.1 Der Matlab - Editor/-Debugger

Neben dem Zeileneditor im MCW, mit dem keine speicherfähigen Programme erzeugt werden können, nutzt man überwiegend den in der integrierten Entwicklungs-Umgebung vorhandenen Matlab-Editor / -Debugger. Ihn öffnet man durch Anklicken des ***Buttons 1* im MCW** (vergleiche Kapitel 2.1) oder wie bei Windows mit ***open*** im ***File-Menue***.
In das Editorfenster (vergleiche das folgende Bild) wird das zu entwickelnde Programm geschrieben, dabei können u. a. die Buttons ***ausschneiden*, *kopieren* und *einfügen*** benutzt werden.



Das Matlab Editor-/Debugger-Fenster

Im ***Pulldown-Menue Tools -> Options -> Editor*** können spezielle Eigenschaften des Editors eingestellt werden. es wird empfohlen ***Syntax Highlighting, Auto indent on return*** (mit ***Auto indent size*** 3 oder 4) (automatisches Einrücken mit 3 oder 4 Leerzeichen bei Betätigung der Returntaste) und bei ***Tab key settings "a tab character"*** mit ***Tab size*** 3 oder 4 (Eine Tabulatorbetätigung führt dann zum Einrücken um 3 oder 4 Zeichen) einzustellen. Die restlichen Einstellungen können beibehalten werden.

Das fertige Programm kann mit ***File speichern*** auf einem Speichermedium (Festplatte, Diskette, usw.) abgelegt werden.
Der besseren Übersicht wegen sollte die Speicherung im Arbeitsordner ***work***, oder einem in diesem vom Benutzer selbst angelegten Unterordner erfolgen. Die Namensergänzung **.m** bei Skript- und Function-Files trägt Matlab selbst ein. Bei Function-Files (sofern der Kopf richtig programmiert ist) trägt Matlab sogar den vollständigen Namen ein.
Mit dem Befehl ***run*** im ***Pulldown-Menue Tools*** kann das Programm gestartet werden. Auftretende Fehler werden im MCW gemeldet. Wie man auf dem

vorangehenden Bild erkennt, lassen sich mehrere Programme öffnen, zwischen denen man mit Hilfe der Reiter am unteren Rand des Editorfensters (***geöffnete Programme***) hin- und herschalten kann.
Alternativ kann man ein geschriebenes Programm auch aus dem MCW starten. Dazu muß sein Name ohne die Ergänzung .m eingegeben und mit der Eingabetaste bestätigt werden. Falls Matlab diese Eingabe mit

"Undefined Function or Variable xxxx"

zurückweist, kann eine falscher Verzeichnispfad die Ursache sein. Mit dem Befehl ***what*** kann der aktuelle Pfad im MCW angezeigt werden. Mit ***cd*** kann dann der Pfad eingestellt werden, in dem sich das zu startende Programm befindet (bei ***cd ..*** muß zwischen cd und dem ersten Punkt unbedingt ein Zwischenraum eingefügt werden). Mit ***dir*** können die Inhalte des aktuellen Arbeitsverzeichnisses angeschaut werden. Die benutzte Syntax ist ähnlich der von DOS. Das gleiche kann grafikgestützt mit dem ***Path Browswer*** (vergleiche Kapitel 2.7.3) durchgeführt werden.

Der Editor bietet neben der Programmerstellung die Möglichkeit des Debuggings. Dazu dient das ***Pulldown-Menue Debug*** im Editorfenster. Die beiden mittleren Befehle ***Set/Clear Breakpoint*** und ***Clear All Breakpoints*** im aufgeklappten Menue finden sich auch in der ***Pushbutton-Leiste*** wieder : ***setzen/löschen von Breakpoints*** und ***löschen aller Breakpoints.***

Eine Debug-Sitzung läuft i.a. wie folgt ab:

Ein Programm, das auch selbst geschriebene Functions durchlaufen kann, wird gestartet. Es bricht mit einer Fehlermeldung ab, daß in einem bestimmten Programm und einer bestimmten Kodezeile ein Fehler aufgetreten ist. Dieses Programm wird vom Anwender in den Editor geladen und mit dem Pulldown-Menue ***Edit -> Go To Line*** der Cursor in die fehlerhaft Zeile gesetzt.
Entweder sieht man den Fehler direkt oder man muß ihn suchen. Dies gelingt häufig durch Setzen eines Breakpoints (Cursor in die gewünschte Zeile, ***Pushbutton setzen/löschen*** betätigen). Es erscheint ein roter Punkt vor dieser Kodezeile, der das Setzen des Breakpoints bestätigt. Läßt man das Programm erneut laufen, hält es an der gekennzeichneten Stelle an und es erscheint ein gelber Pfeil. Die Kodezeile, wo der Breakpoint gesetzt ist, wird selbst nicht mehr berechnet.
Egal ob sich der Breakpoint in einem Skrip- oder Function-File befindet, können jetzt die Inhalte der Variablen betrachtet werden. Dies geschieht durch Markierung der gewünschten Variablen (click and drag), Anklicken der markierten Variablen mit der rechten Maustaste und Wahl von ***Evaluate Selection*** in dem sich öffnenden

Befehlsfenster. Der Variableninhalt wird dann im MCW angezeigt. Das gleich kann man durch Eingabe des Variablennamens (CR) im MCW erreichen.
Das MCW macht durch Ausgabe von **K>>** an Stelle von **>>** deutlich, daß sich das Programm im Debug-Modus befindet.
Die weiteren Befehle, die sich ***Pulldown-Menue Debug*** des Editorfensters befinden haben folgende Bedeutung:

Continue:	Der Programmlauf wird fortgesetzt bis zum nächsten oder wiedererreichen des gleichen Breakpoints.
Single Step:	Die aktuelle Kodezeile (Pfeil)wird berechnet. Der Befehl kann fortgesetzt angewendet werden.
Step in:	Wie ***Single Step***, verzweigt aber in aufgerufene Functions.
Quit Debugging:	Beendet eine Debug-Sitzung allerdings ohne die Breakpoints zu löschen. Das muß vorher mit ***Clear All Breakpoints*** oder dem entsprechenden Pushbutton vorgenommen werden.
Stop if ... Error, Warning, NaN	Das Programm hält ohne Setzen von Breakpoints bei Auftreten eines der drei Ereignisse an, ohne in den Debug-Modus zu gehen.

Etwas aufwendiger wird eine Debug-Sitzung wenn kein Syntax-Fehler auftritt, sondern logische Fehler debugged werden sollen. Durch Einsatz der obigen Werkzeuge gelingt aber auch das.

Eine Debug-Sitzung wird beendet mit dem ***Pushbutton Clear All Breakpoints*** und ***Quit Debugging*** im ***Pulldown-Menue Debug*** im Editorfenster.

Auch ohne Einsatz des Debuggers ist es mit den folgenden Hinweisen oft möglich Programmfehler nur im MCW aufzudecken:

- Falls ein Programm in eine Endlosschleife gerät, oder die Programmausführungszeit zu langsam erscheint, kann mit der Tastenkombination "**Strg+c**" der Programmlauf unterbrochen werden.
- Durch Weglassen des Semikolons am Ende einer Befehlszeile werden die mit diesem Befehl erzeugten Variablen während des Programmlaufs mit ihrem Namen auf dem Bildschirm angezeigt. Die Analyse dieser Ausgaben führt häufig zur Aufdeckung von Fehlern.
- Durch "Wegkommentierung" von Befehlszeilen mit Hilfe des Kommentarzeichens "**%**" kann der Einfluß fehlender Befehle getestet werden.
- Wird nach Ablauf eines Programms oder nach einem durch eine Matlab-Fehlermeldung hervorgerufenen Programmabbruch der Befehl "**whos**"

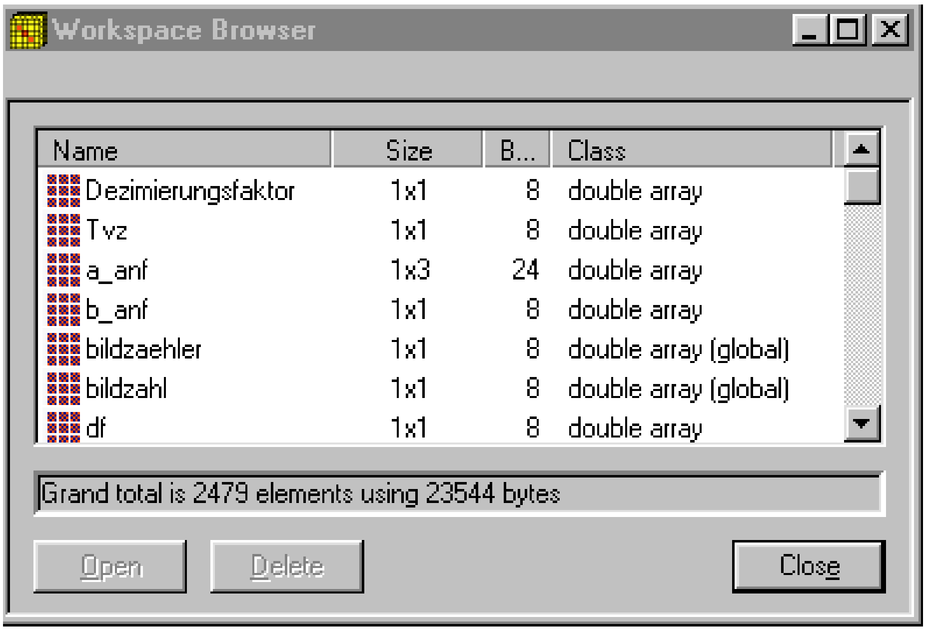
eingegeben (oder der Button 7 im MCW betätigt), zeigt Matlab alle im Variablenspeicher vorliegenden Variablen mit ihrem Namen und ihrer Matrix-Dimension an:

Variablenname   size   m by n   ...

- Dabei ist m die Anzahl der Zeilen und n die Anzahl der Spalten der Variablenmatrix mit dem Namen "Variablenname". Hier werden oft Inkompatibilitäten zwischen verschieden großen Vektoren und Matrizen offenbar, die miteinander verknüpft werden sollen. Beim Arbeiten mit "**whos**" sollte vor dem Start des zu testenden Programms der Matlab-Arbeitsspeicher mit dem Befehl "**clear**" gelöscht werden.

2.7.2 Der Matlab Workspace-Browser

Der Workspace-Browser öffnet sich nach Anklicken des Button 7 im MCW .

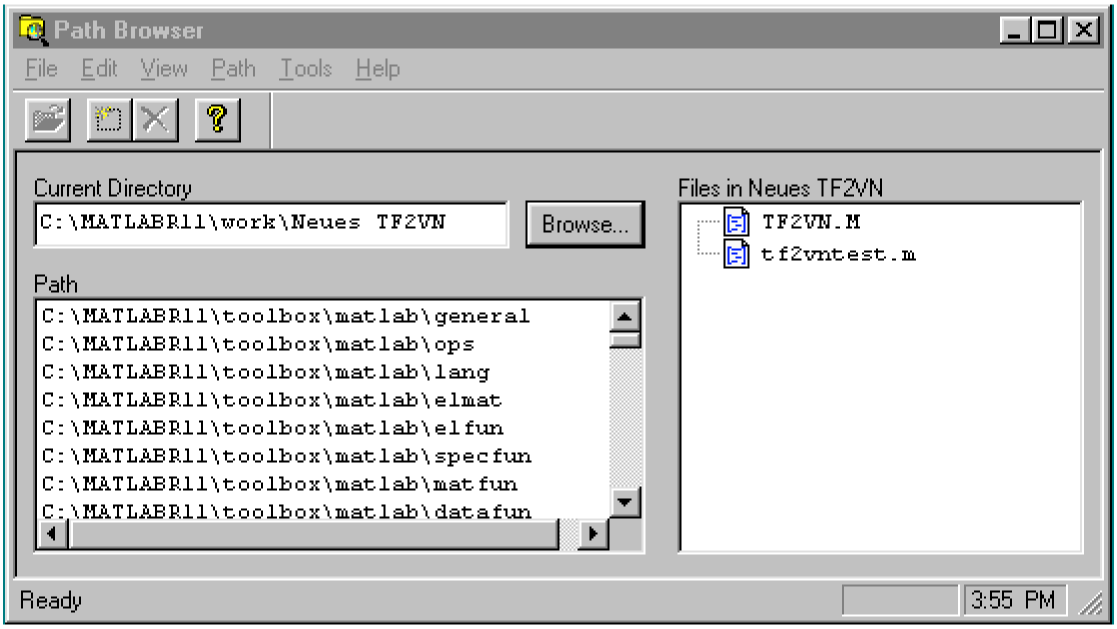


Der Matlab Workspace-Browser

Hier werden in einer übersichtlichen Form alle im Matlab-Arbeitsspeicher befindlichen Variablen mit Angabe des Variablennamens, ihrer spezifischen Größe, ihres Platzbedarfs in Bytes und um welche Datentyp/ welche Datenstruktur es sich handelt, dargestellt. Es handelt sich hierbei um ein grafikgestütztes Pendant zum Befehl "**whos**". Die Variablen können mit dem Editor betrachtet und geändert werden (Doppelklick auf den Variablennamen) und einzeln und in Gruppen gelöscht werden.

2.7.3 Der Matlab Path-Browser

Der Matlab Path-Browser (Button 8 im MCW) dient zum Betrachten und Ändern des Arbeitsverzeichnisses und zur Festlegung von Suchpfaden (siehe folgendes Bild).
Nach Betätigung des Buttons ***Browse*** kann das Arbeitsverzeichnis ausgewählt werden. Der ausgewählte Pfad erscheint dann in der Zeile ***Current Directory*** und die darin enthaltenen m- und mat-Files in rechten Rubrik ***Files in XXX***. Nach dem Wegklicken des Path-Browser-Fensters ist das Arbeitsverzeichnis eingestellt.



Der Path-Browser von Matlab

Alle in der linken unteren Rubrik ***Path*** aufgeführten Pfade sind sogenannte Matlab-Suchpfade, die von Matlab bei der Abarbeitung eines Programms nach darin benutzten Befehlen durchsucht werden. Wird in einem Programm ein Befehl benutzt, auf den kein Suchpfad führt, gibt Matlab die Fehlermeldung

"Undefined Function or Variable xxxx"

aus. Mittels ***Browse*** kann dann der den Befehl enthaltene Pfad als ***Current Directory*** ausgewählt und über das ***Pulldown-Menue Path -> Add to Path*** den Suchpfaden (***Path***) zugefügt werden.

3 Matlabfunktionen für Problemstellungen aus der Systemtheorie und Regelungstechnik

Im folgenden werden einige Matlab-Funktionen aus dem Grundbefehlssatz und der Control-System Toolbox zur Lösung von Problemstellungen aus der Systemtheorie und Regelungstechnik vorgestellt.
Im Rahmen dieses Skriptes werden wir uns auf lineare, zeitinvariante Eingrößen-systeme konzentrieren. Dabei werden kontinuierliche und zeitdiskrete Systeme berücksichtigt.

3.1 Kontinuierliche Systeme

3.1.1 Systemgenerierung

Mathematische Modelle von Übertragungssystemen können unter Matlab als Zustandsmodell im Zeitbereich oder als Übertragungsfunktion im s-Bereich eingegeben werden.

3.1.1.1 Eingabe von Zustandsmodellen

Liegt ein mathematisches Modell in Zustandsform vor

$$\begin{aligned} \dot{\underline{x}}(t) &= \underline{A}\,\underline{x}(t) + \underline{b}\,u(t) \\ y(t) &= \underline{c}'\,\underline{x}(t) + d\,u(t) \end{aligned}$$

, zum Beispiel

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} = \begin{bmatrix} -2 & 2 \\ 0{,}5 & -2 \end{bmatrix} \cdot \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 1{,}5 \end{bmatrix} \cdot u(t)$$

$$y(t) = \begin{bmatrix} 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix}$$

müssen die Matrizen, Vektoren und Skalare $\underline{A}$, $\underline{b}$, $\underline{c}'$ und d wie folgt nach den Regeln der Eingabe von Matrizen, bzw. Vektoren eingegeben werden:

```
A=[-2, 2; 0.5,-2];    b=[0; 1.5];    c=[0, 1];    d=0;
```

Da die in den folgenden Kapiteln beschriebenen Umformungs- und Analysebefehle für Systeme mit einem sogenannten *LTI-Objekt (Linear-Time-Invariant-Objekt)* zur Systembeschreibung parametriert werden, muß aus den Systemmatrizen A, b, c, d ein Zustandsmodell-Objekt zstm = **ss** (A, b, c, d);

erzeugt werden. Durch Eingabe des LTI-Objekt-Namens "zstm (CR)" oder weglassen des Semikolons im obigen Befehl gibt Matlab in einer anschaulichen Form den Inhalt und die Art (Continous-time model) des Zustandsmodell-Objekts aus:

```
a =
                x1    x2
         x1     -2     2
         x2    0.5    -2
b =
                u1
         x1      0
         x2    1.5
c =
                x1    x2
         y1      0     1
d =
                u1
         y1      0

Continuous-time model.
```

Um auf die in "zstm" (verschlüsselt) gespeicherten Parametermatrizen des System oder ihre Elemente zugreifen zu können, wird der Befehl

[A, b,c,d] = **ssdata** (zstm);

benutzt.

Ein Zustandsmodell-Objekt heißen im Matlab-Arbeitsspeicher "ss objekt". Bei der Eingabe von **whos (CR)** gibt Matlab für das obige Zustandsmodell-Objekt "zstm" folgende Information aus:

```
>>whos
  Name         Size          Bytes  Class

   ⋮            ⋮               ⋮      ⋮

  zstm         1x1            2844  ss object

Grand total is 56 elements using 2988 bytes
```

Das folgende Programm "zustm.m" demonstriert diese Zusammenhänge:

```
% Programm "zustm.m"
% Erzeugung und Nutzung eines Zustandsmodellobjekts

clear, home, close all

% Parametermatrizen des Zustandsmodells
%
A=[-2,2;0.5,-2];  % Systemmatrix
```

```
b=[0;1.5];          % Eingangsvektor
c=[0,1];            % Ausgangsvektor
d=0;                % Durchgangsfaktor

% Bildung des LTI-Objekts: Zustansmodellobjekt "zstm"
%
zstm=ss(A,b,c,d);

% Anschauliche Ausgabe des Zustandsmodellobjekts
% auf dem MCW
%
zstm

% Zugriff auf die Parametermatrizen im
% Zustandsmodellobjekt
%
[A_zmo, b_zmo, c_zmo, d_zmo]=ssdata(zstm);
```

3.1.1.2 Eingabe von Übertragungsfunktionen

Übertragungsfunktionen sind gebrochen rationale Funktionen d.h. der Quotient zweier Polynome. Diese werden , wie im Kapitel 2.3.5 beschrieben , zunächst eingegeben: Beispiel:

$$G(s) = \frac{4s^2 + 2s + 1}{3s^2 + 10s + 100},$$

daraus folgt die Matlab-Eingabe:

$$\text{zaehler} = [4,2,1]; \qquad \text{nenner} = [3,10,100];.$$

Die Umformung in ein LTI-Objekt erfolgt mit dem Befehl

uetf = **tf** (zaehler, nenner);

Wiederum durch Eingabe des LTI-Objektnamens "uetf (CR)" oder weglassen des Semikolons im obigen Befehl wird die Übertragungsfunktion in anschaulicher Weise auf dem MCW ausgegeben:

```
Transfer function:
4 s^2 + 2 s + 1
------------------ .
3 s^2 + 10 s + 100
```

Um auf das Zähler- und das Nennerpolynom innerhalb des Übertragungsfunktions-Objekts zugreifen zu können, wird zunächst der Befehl

[z,n] = **tfdata** (uetf);

angewendet. "z" und "n" sind allerdings keine Vektoren (wie A, b, c, und d bei der Verwendung von **ssdata** bei Zustandsmodellen) sondern Cell-Arrays. Deshalb gibt der Befehl **tfdata** nur die Struktur der Cell-Arrays von z und n aus (wenn man **whos** (CR) eingibt oder das Semikolon hinter dem obigen Befehl wegläßt).
Bei den von uns ausschließlich betrachteten SISO-Systemen (Single-Input / Single-Output-Systeme, Eingrößensysteme) handelt es sich immer um Cell-Arrays mit nur einer Zelle z{1,1}, n{1,1}. Diese Inhalte kann man einer Variablen zuweisen:

zaehler_uetf = z**{**1,1**}**;
nenner_uetf = n**{**1,1**}**;

und verfügt damit über die Koeffizienten des Zähler- und Nennerpolynoms der zugrunde liegenden Übertragungsfunktion.

Das folgende Programm "uebtf.m" demonstriert diese Zusammenhänge:

```
% Programm "uebtf.m"
% Erzeugung und Nutzung eines Übertragungsfunktions-Objekts

clear, home, close all

% Polynome der Übertragungsfunktion
zaehler=[4,2,1];
nenner=[3,10,100];

% Bildung des LTI-Objekt: Übertragungfunktions-Objekt "uetf"
uetf=tf(zaehler, nenner);

% Anschauliche Ausgabe des Übertragungsfunktionsobjekts
% auf dem MCW
%
uetf

% Zugriff auf das Zähler- und Nennerpolynom im
% Übertragungsfunktionsobjekt
%
[z, n]=tfdata(uetf)
% z und n sind Cell-Arrays (siehe MCW). Um aus diesen
% die Zähler- und Nenner-Polynome zu extrahieren, muß
% mit geschweiften Klammern indiziert werden
%
zaehler_uetf=z{1,1}
nenner_uetf=n{1,1}
```

Die Eingabeparameter-Listen der Befehle **tf** und **ss** können noch um weitere Elemente der Form

zstm = **ss** (A, b, c, d, *'Propertyname', Propertyvalue, ...*) bzw.
uetf = **tf** (zaehler, nenner, *'Propertyname', Propertyvalue, ...*)

erweitert werden. Wir wollen mit dieser Befehlserweiterung gegebenenfalls vorhandene Totzeiten in die Systemmodelle integrieren. Der "*Propertyname*" heißt in diesem Falle **ioDelay**, der *Porpertyvalue* muß die Dauer der Totzeit in Sekunden enthalten.

Es gibt noch andere "Properties", die in die obige Liste aufgenommen werden können. Sie können mit **ltiprops (CR)** im MCW nachgelesen werden.

3.1.2 Umformung von Systembeschreibungen

Alle wichtigen Systembeschreibungsformen (Zustandsmodelle, Übertragungsfunktionen in ihren verschiedenen Schreibweisen) lassen sich mit einfachen Matlab-Befehlen ineinander umrechnen.

3.1.2.1 Umformungen zwischen Zeit- und s-Bereich

Mit den schon eingeführten Befehlen **tf** und **ss** lassen sich aus einem beliebigen LTI-Objekt, das in der Matlab-Literatur häufig mit "sys" abgekürzt wird. ("sys" kann also ein Zustandsmodell- oder Übertragungsfunktions-Objekt sein) jedes andere LTI-Objekt berechnen:

uetf = **tf** (sys); % wandelt in ein Übertragungsfunktions-Objekt
zstm = **ss** (sys); % wandelt in ein Zustandsmodell-Objekt

(Es sei an dieser Stelle noch einmal ausdrücklich darauf hingewiesen, daß bei der Umwandlung einer Übertragungsfunktion in ein Zustandsmodell, dieses eine besondere Form annimmt, die nicht die physikalische Struktur des Systems widerspiegelt. Es wird eine sog. kanonische Form berechnet, in unserem Falle die "Regler-Normalform". Siehe dazu /1/.)

3.1.2.2 Umformungen innerhalb des s-Bereichs

Liegen die Vektoren der Polynomkoeffizienten von Zähler (zaehler) und Nenner (nenner) der Übertragungsfunktion in Polynomform (Matlab-Abkürzung tf: Transfer Function) fest, lassen sich daraus alle anderen Schreibweisen

- Produktform (Matlab: zp (zero / pole))
- Partialbruchform (Matlab: Residue)
- V-Normalform (vn)

mit den folgenden Befehlen berechnen

- **Berechnung der Produktform**

  [null, pol, k] = **tf2zp** (zaehler, nenner);

  Dabei sind "null" die Nullstellen, "pol" die Polstellen und "k" der konstante Faktor "K" der Produktform ("zero-pol"-Form) der Übertragungsfunktion

$$G(s) = K \cdot \frac{(s - \text{null}(1)) \cdot (s - \text{null}(2)) \cdot \cdots \cdot (s - \text{null}(m))}{(s - \text{pol}(1)) \cdot (s - \text{pol}(2)) \cdot \cdots \cdot (s - \text{pol}(n))} \, .$$

  Zur umgekehrten Berechnung der Polynomform aus der Produktform dient der Befehl [zaehler, nenner] = **zp2tf** (null,pol,k);

  Bemerkung zur Nomenklatur der Matlab-Befehle
  In der Befehelsmnemonik wird die "2" (two) mit dem gleichklingenden "to" interpretiert. So gelesen ergibt der obige Befehl einen Sinn:
  tf**2**zp = Transfer-Function **to** zero-pole (-Umformung)

- **Berechnung der Partialbruchform**

  [r, p, k] = **residue** (zaehler, nenner)

  Der Befehl verarbeitet einfache, mehrfache und komplexe Nullstellen im Zähler und im Nenner. Wegen der Komplexität der entstehenden Lösung sei auf die Erklärung im help-Text hingewiesen ("help residue"). Der Warnhinweis am Ende der Erläuterungen ist besonders zu beachten.
  Zur umgekehrten Berechnung der Polynomform aus der Partialbruchform dient der Befehl [zaehler, nenner] = **residue** (r, p, k);.

- **Berechnung der faktorisierten V-Normalform** /1/

  Leider wird diese Umformung nicht von Matlab unterstützt. Es wird deshalb ein vom Autor entwickelter Befehl angegeben :

  tf2vn (zaehler, nenner);

  Der Befehl hat keine linksseitigen Argumente. Das Ergebnis, die faktorisierte V-Normalform, wird nur auf dem MCW dargestellt.

Beispiel: Die Übertragungsfunktion in Polynomform

$$G(s) = \frac{100s^3 + 60s^2 + 105s + 50}{100s^8 + 230s^7 + 962s^6 + 1074s^5 + 1858s^4 + 496s^3 + 32s^2}$$

wird mit folgendem Matlab-Programm in die V-Normalform gewandelt:

```
z=[100, 60,  105, 50];
n=[100, 230, 962, 1074, 1858, 496, 32, 0, 0];
tf2vn(z,n)
```

wobei folgendes Ergebnis ausgegeben wird:

```
Verstärkungsfaktor: V = 1.5625

Globalverhalten:    k = 2


Zählerdynamik: Z(s)=

(1 + 2*0.05*1*s + 1*s^2)
(1 + 2*s)

Nennerdynamik: N(s)=

(1 + 2*0.25*0.5*s + 0.25*s^2)
(1 + 2*0.25*0.5*s + 0.25*s^2)
(1 + 5*s)
(1 + 10*s)
```

Damit wird folgende V-Normalform beschrieben:

$$G(s) = 1.5625 \cdot \frac{1}{s^2} \cdot \frac{\left(1 + 2 \cdot 0.05 \cdot s + s^2\right) \cdot (1 + 2 \cdot s)}{\left(1 + 2 \cdot 0.25 \cdot 0.5 \cdot s + 0.25 \cdot s^2\right)^2 \cdot (1 + 5 \cdot s) \cdot (1 + 10 \cdot s)}$$

3.1.3 Systemanalyse von Übertragungssystemen

Matlab bietet eine Reihe von Analysemethoden für Übertragungssysteme an:

- Im Zeitbereich

Berechnung von Systemantworten und Zustandsgrößenverläufen bei verschiedenen Eingangssignalformen und/oder aus Anfangszuständen der Zustandsgrößen. Dieser Vorgang wird in der Literatur häufig als <u>Systemsimulation</u> beschrieben.

- Im s-Bereich:

Analyse der Übertragungsfunktion, speziell der V-Normalform. Berechnung der Pol-/Nullstellenverteilung einer Übertragungsfunktion.

- Im Frequenzbereich:

Berechnung der Ortskurve oder des Bodediagramms des Frequenzganges eines Übertragungssystems

Alle Matlab-Analysebefehle arbeiten mit Systembeschreibungen in Form eines LTI-Objekts. Dies kann ein Zustandsmodell-Objekt (erzeugt mit **ss**) oder ein Übertragungsfunktions-Objekt (erzeugt mit **tf**) sein. Es wird im folgenden "sys" genannt.
Es sei bemerkt, daß eine Zeitbereichsanalyse, z.B. die Berechnung einer Sprungantwort, auf der Basis eines Übertragungfunktions-Objekts eine relativ aufwendige Umrechnung des Systems vom Bild- in den Zeitbereich erfordert (vergleiche /1/).
Genau so muß eine Umrechnung vom Zeit- in den Bildbereich erfolgen, wenn von einer Systembeschreibung in Form eines Zustandsmodell-Objekts z.B. ein Pol-/Nullstellenplan im Bildbereich berechnet werden soll.
Alle diese Umrechnungen nimmt Matlab vor, ohne das der Benutzer davon etwas bemerkt.

3.1.3.1 Systemanalyse im Zeitbereich

Bei allen Matlab-Befehlen zur Berechnung von Systemantworten im Zeitbereich wird eine Zeitskala benötigt, über der die Systemantwort berechnet werden soll. Sie wird mit Hilfe des schon bekannten Befehls

t = 0 : dt : te;	dt:	Auflösung der Zeitachse
	te:	Endzeitpunkt

erstellt. Die Rechenschrittweite dt sollte kleiner/gleich 1/100 der kleinsten Zeitkonstante des zu simulierenden Systems gewählt werden (V-Normalform berechnen):

$$dt \le \frac{1}{100} \cdot T_{min}; \qquad T_{min}: \text{ kleinste System-Zeitkonstante.}$$

Sinusförmige Erregungen dürfen dann eine maximale Frequenz von

$$f_{Err(max)} = 0,5 \cdot \frac{1}{dt}$$

haben, um das Shannonsche Abtasttheorem nicht zu verletzen.

- **Berechnung der Gewichtsfunktion**

Die Antwort eines Übertragungsgliedes auf einen Dirac-Stoß

$$u(t) = \begin{cases} \infty & \text{für} \quad t = 0 \\ 0 & \text{für} \quad t \neq 0 \end{cases}$$

ist die sog. Gewichtsfunktion g(t) des Systems. Sie berechnet sich unter Matlab wie folgt:

g = **impulse** (sys, t);

Wobei "g" der Vektor der Gewichtsfunktionswerte ist. Er hat die gleiche Länge wie der Zeitvektor "t". "sys" beschreibt das zu analysierende System.

- **Berechnung der Sprungantwort**

Die Antwort eines Übertragungssystems auf einen Sprung der Eingangsgröße mit der Amplitude A

$$u(t) = A \cdot \sigma(t) = \begin{cases} A & \text{für} \quad t \geq 0 \\ 0 & \text{für} \quad t < 0 \end{cases}$$

nennt man die Sprungantwort oder Übergangsfunktion eines Übertragungssystems. Sie berechnet sich unter Matlab wie folgt:

h = A***step** (sys,t);

wobei "h" der Vektor der Sprungantwort mit der Länge von "t" ist. Die restliche Parametrierung entspricht der von "impulse". "A" ist die Amplitude des Sprunges.

- **Systemantworten auf beliebige Eingangssignale**

Die Systemantwort auf ein beliebiges Eingangssignal wird mit dem Befehl

y = **lsim** (sys, u, t, x0);

berechnet. Der Vektor "u" ist der Vektor der Kurvenform des Eingangssignals, der vor dem Aufruf von "lsim" festgelegt (berechnet) werden muß. <u>Der Vektor von "u" muß die gleiche Länge wie der Vektor "t" haben</u>. x0 ist der (Spalten-) Vektor der Anfangszustände der Zustandsgrößen. Er kann weggelassen werden, dann werden die Anfangszustände auf Null gesetzt.
In bestimmten Situationen ist auch die Kenntnis des Verlaufs der Zustandsgrößen $\underline{x}(t)$ notwendig. Diese können dann mittels der Parametrierung

[y,ts, x] = **lsim** (sys, u, t, x0);

berechnet werden. Dies ist allerdings nur dann sinnvoll, wenn die Parametermatrizen aus einer mathematisch/physikalischen Modellbildung hervorgegangen sind (vergleiche /1/).
In der vorliegenden Matlab-Version (5.3, R11) kann der Rückgabe-Zeitvektor "ts" eine feinere Auflösung haben als der vorgegebene Zeitvektor "t", um ggf. auftretende Schwingungen zwischen den Abtastzeitpunkten darstellen zu können. Nähere Einzelheiten siehe "help lsim".

Folgende Befehle können u. a. zur Erzeugung spezieller Eingangssignalformen benutzt werden (nähere Einzelheiten siehe "help ..."):

Sinus, Kosinus:	**sin, cos**
Rechteckschwingung:	**square**
Sägezahnschwingung:	**sawtooth**
(verzögerte) Sprungfunktion:	**stepfun**
Rauschsignale:	**randn**

3.1.3.2 Systemanalyse im s-Bereich

Systemanalysen im s-Bereich lassen sich an der Struktur der Übertragungsfunktion, speziell an der sog. "faktorisierten V-Normalform" (siehe weiter vorn) und an der Pol-/Nullstellenverteilung vornehmen. Mit Hilfe des Befehls

pzmap (sys);

wird eine Grafik der Pol-/Nullstellenverteilung in der s-Ebene erzeugt. "zaehler" und "nenner" sind wieder die Koeffizientenvektor von Zähler und Nenner der Übertragungsfunktion in Polynomform. Die Lage der Nullstellen wird durch kleine Kreise, die der Polstellenlagen durch kleine Kreuze gekennzeichnet.

3.1.3.3 Systemanalyse im Frequenzbereich

Systemanalysen im Frequenzbereich lassen sich an der Ortskurve und vorzugsweise dem Bodediagramm des Frequenzganges vornehmen.

- **Die Ortskurve des Frequezganges**

Der Matlab-Befehl

[re,im] = **nyquist** (sys, w);

berechnet den Realteil "re" und den Imaginärteil "im" eines Übertragungssystems mit der Übertragungsfunktion G(s) (sys) für die im Vektor "w" spezifizierte ω-Folge. Der Frequenz-Vektor "w" kann linear w = wa **:** dw **:** we; oder logarithmisch w=**logspace**(d1,d2,n); geteilt sein. Die Teilung (dw bzw. n) wird zweckmäßig experimentell ermittelt. Ziel ist ein möglichst glatter Verlauf der Kurve ohne "Ecken".

- **Das Bodediagramm des Frequenzganges**

Der Befehl

[betrag, phas] = **bode** (sys, w);

berechnet den Betrag "betrag" und die Phase "phas" eines Übertragungssystems mit der Übertragungsfunktion G(s) (sys) für die im Vektor "w" spezifizierte ω-Folge. Weil der Befehl **bode** auch auf Mehrgrößensysteme anwendbar ist, sind die Betrags- und Phasenverläufe ("betrag" und "phas") in dreidimensionalen Matrizen [mxnxq] untergebracht. Um daraus einfache Vektoren zu erzeugen, die nur den Betrags- und den Phasenverlauf enthalten, muß der Befehl **reshape** angewendet werden. Da wir uns nicht mit Arrays der Dimension >2 auseinandersetzen wollen, soll die Funktion des Befehls nicht näher erläutert werden. Die "reshape"-Befehle müssen nur in der folgenden Form im Kode zur Darstellung eines Bodediagramms auftauchen (siehe die Kodesequenz weiter unten). Anstelle von “phase“ wird abgekürzt “phas“ benutzt, weil **phase** ein Matlab-Befehl ist.

Mit dem Befehl

btr_db = 20***log10**(betrag);

lassen sich dann die Betragswerte in Dezibel (dB) umrechnen. Der ω-Vektor "w" wird zweckmäßig mit dem schon bekannten Befehl **logspace** erzeugt.

Die folgende Kodesequenz berechnet das Bodediagramm eines gegebenen Übertragungssystems (sys) und stellt es grafisch in zwei übereinanderlegenden Koordinatensystemen über der Frequenz ω [1/sek] dar, und zwar von $w_{Anfang} = 10^a$ bis $\omega_{Ende} = 10^e$. In der oberen Bildhälfte befindet sich die Betrags- in der unteren die Phasenkennlinie:

```
w=logspace (a,e,600);       % logarithmische Frequenzachse.
[betrag, phas]=bode(sys,w);% Berechnung des Bodediagramms.

betrag=reshape(betrag,[1,length(w)]);
phas=reshape(phas,[1,length(w)]);
                            % Wandlung der 3D Arrays "btr"
```

```
                              % und "ph" in Vektoren.

b_dB=20*log10(betrag);        % Umrechnung des Betrages in dB.
subplot(2,1,1);               % Plot in obere Bildhälfte.
semilogx(w,b_dB);             % Plotbefehl bei logaritmischer
                              % Abszisse.
grid on                       % Gitternetz über Grafik.
title('Betragskennlinie')     % Titelbeschriftung (obere Grafik).
ylabel('dB')                  % Ordinatenbeschriftung ( " ).
subplot(2,1,2);               % Plot in untere Bildhälfte.
semilogx(w,phas);             % Plotbefehl bei logaritmische
                              % Abszisse.
grid on                       % Gitternetz über Grafik.
title('Pasenkennlinie')       % Titelbeschriftung(untere Grafik).
ylabel('Grad');               % Ordinatenbeschriftung (   "   ).
xlabel('Omega [1/sek]')       % Abszissenbeschriftung.
```

3.2 Zeitdiskrete Systeme

Matlab erlaubt auch eine umfassende Analyse und Synthese zeitdiskreter Systeme. Wir beschränken uns auch hier auf lineare, zeitinvariante Eingrößensysteme.

3.2.1 Systemgenerierung

Zeitdiskrete Systeme können unter Matlab auch in Form von (z-)Übertragungsfunktionen und Zustandsmodellen eingegeben werden. Die Eingabeform als rekursive Differenzengleichung wird von Matlab nicht unterstützt und Zustandsmodelle zeitdiskreter Übertragungssysteme wurden in /1/ nicht behandelt. Bei der direkten Eingabe zeitdiskreter Systembeschreibungen wollen wir uns deshalb auf z-Übertragungsfunktionen beschränken.

Dies bedeutet auch für Zeitbereichsanalysen keine wesentliche Einschränkung, da die Analysebefehle für den Zeitbereich auch mit Übertragungsfunktions-Parametern ("zaehler", "nenner" der z-Übertragungsfunktion) parametriert werden können (vergleiche die Ausführungen zu Beginn des Kapitels 3.1.3)

Die Eingabe der Übertragungsfunktionen erfolgt wieder durch die Vektoren der Zähler- und Nennerpolynom-Koeffizienten in der Reihenfolge fallender positiver Potenzen von z:

$$G(z) = \frac{b_m z^m + b_{m-1} z^{m-1} + \cdots + b_2 z^2 + b_1 z + b_0}{a_n z^n + a_{n-1} z^{n-1} + \cdots + a_2 z^2 + a_1 z + a_0}$$

Daraus folgt die Matlabeingabe

$$\text{zaehler} = \left[b_m, b_{m-1}, \cdots, b_2, b_1, b_0\right]; \qquad \text{nenner} = \left[a_n, a_{n-1}, \cdots, a_2, a_1, a_0\right]$$

> **Achtung:** Die Eingabe der Koeffizienten nach fallenden <u>negativen</u> Potenzen von z (wie die z-Übertragungsfunktion häufig geschrieben wird) führt zu Fehlern!

Die Umformung in eine LTI-Objekt erfolgt wieder mit dem Befehl **tf**, allerdings in einer etwas anderen Parametrierung

uetf = **tf** (zaehler, nenner, T); T: Abtastzeit des Systems

Auch hier kann durch Eingabe des LTI-Objekt-Namens "uetf (CR)" oder weglassen des Semikolons im obigen Befehl die eingegebene Übertragungsfunktion in anschaulicher Bruchstrich-Darstellung auf dem MCW ausgegeben werden.
Die Eingabeparameter-Liste kann auch um weitere Elemente der Form

uetf = **tf** (zaehler, nenner, T,*'Propertyname', Propertyvalue, ...*)

erweitere werden. Die Funktionalität der Erweiterung ist genau die gleiche wie bei kontinuierlichen LTI-Objekten. Die einstellbaren "Properties" können auch mit "ltiprops (CR)" auf dem MCW nachgelesen werden.

Eine zweite Möglichkeit ein zeitdiskretes System zu erzeugen, besteht darin, ein vorliegendes kontinuierliches System mittels einer Diskretisierungs-Transformation zu diskretisieren.

3.2.2 Diskretisierungs-Transformationen

Steuert man ein kontinuierliches System über einen D/A-Wandler (Halteglied) an und entnimmt ihm zu den Abtastzeitpunkten (z.B. mittels eines A/D-Wandlers) seine Ausgangsgröße (Abtaster), kann das Gesamtsystem (einschließlich der Wandler) als zeitdiskretes System aufgefaßt werden. Matlab unterstützt diesen Diskretisierungs-Vorgang, der z.B. beim Entwurf digitaler Filter (einfache Ansätze werden in /1/ beschrieben) und zeitdiskreter Regler /2/ benutzt wird, mit dem Befehl **c2d** ("<u>C</u>ontinuous to <u>D</u>iscrete"):

[sys_d] = **c2d** (sys_k, T,**'**methode**'**); .

dabei bedeuten

sys_k: LTI-Objekt des kontinuierlichen Systems,
sys_d: LTI-Objekt des zeitdiskreten Systems,
T: Abtastzeit.

Als "methode" kann eine Diskretisierungstransformation aus der folgenden (nicht vollständigen) Liste gewählt werden (vergleiche /1/):

'**zoh**': zero-order-hold, Sprunginvarianz-Transformation

$$G(z) = \frac{z-1}{z} \cdot \mathscr{Z}\left\{\frac{G(s)}{s}\right\};$$

'**foh**': first-order-hold, Rampeninvarianz-Transformation

$$G(z) = \frac{(z-1)^2}{z} \cdot \mathscr{Z}\left\{\frac{G(s)}{s^2}\right\};$$

'**tustin**': Tustinsche Näherung (auch Bilinear-Transformation)

$$G(z) = G(s)\Big|_{s = \frac{2}{T} \cdot \frac{z-1}{z+1}} \quad .$$

Zur umgekehrten Wandlung eines zeitdiskreten Systems in ein kontinuierliches stellt Matlab den Befehl [sys_k] = **d2c** (sys_d, 'methode') zur Verfügung.

3.2.3 Umformungen zeitdiskreter Übertragungsfunktionen

Mit Hilfe der schon bei den kontinuierlichen Systemen eingeführten Befehlen

[null, pol, K] = **tf2zp** (zaehler, nenner);
[r, p, k] = **residue** (zaehler, nenner);

sowie ihrer Umkehrungen

[zaehler, nenner] = **zp2tf** (null, pol, K);
[zaehler, nenner] = **residue** (r,p,k);

können verschiedene Darstellungsformen der z-Übertragungsfunktion berechnet werden. Die V-Normalform ist in der in /1/ beschriebenen Form bei zeitdiskreten Systemen nicht definiert.

3.2.4 Systemanalyse zeitdiskreter Systeme

Für die Systemanalyse zeitdiskreter Systeme werden die gleichen Matlab-Befehle wie bei kontinuierlichen Systemen benutzt.
Es muß lediglich sichergestellt sein, daß es sich bei dem verwendeten System "sys", um ein zeitdiskretes System handelt , das nach den Methoden des Kapitels 3.2.1 oder 3.2.2 generiert wurde.

3.2.4.1 Systemanalyse im Zeitbereich

Auch die Zeitachse

t = 0 **:** T **:** tend; (tend: Simulations-Endzeitpunkt)

wird bei zeitdiskreten Systemen anders erzeugt. Im Gegensatz zur Zeitachse für die Simulation kontinuierlicher Systeme ist die Schrittweite T der Zeitachse nicht eine möglichst kleine, sondern genau die **Abtastzeit T**, die zur Diskretisierung des zu simulierenden Systems verwendet wurde.

Der Befehl

g = **impulse**(sys, t);

berechnet die Impulsantwort (Gewichtsfolge) eines zeitdiskreten Systems, d.h. die Systemantwort auf die Eingangserregung

$$u(k) = \begin{cases} 1 & \text{für} \quad k = 0 \\ 0 & \text{für} \quad k \neq 0 \end{cases} \quad .$$

Der Befehl

h = **step**(sys, t);

berechnet die Einheitssprungantwort eines zeitdiskreten Systems, d.h. die Systemantwort auf die Eingangsfolge

$$u(k) = \begin{cases} 1 & \text{für} \quad k \geq 0 \\ 0 & \text{für} \quad k < 0 \end{cases} \quad .$$

Der Befehl

y = **lsim**(sys, u, t);

berechnet die Systemantwort eines zeitdiskreten Systems auf eine beliebige Eingangsfolge u, die vor dem Aufruf von dlsim bereitgestellt werden muß. U muß die gleiche Länge wie die Zeitachse T haben. Der berechnete Ausgangssignal-Vektor y hat die gleiche Länge wie die Eingangssignal-Vektor u.

Zusatzbemerkung: Will man die berechneten Ausgangssignalformen grafisch darstellen, bediente man sich bei kontinuierlichen Systemen des Befehl "plot", der die berechneten Ausgangssignalpunkte in Form von Polygonzügen darstellte. Um bei

Eingangs- und Ausgangssignalen zeitdiskreter Systeme deutlich zu machen, daß es sich um zeitdiskrete Signale handelt, benutzt man an Stelle von "plot" häufig die Befehle

stairs oder **stem .**

"stairs" zeichnet Signale in Form von Treppenfunktionen (häufig Ausgangssignal zeitdiskreter Systeme), "stem" gibt Einzelimpulse mit einem kleinen Kreis auf der Spitze aus (Eingangssignal zeitdiskreter Systeme).

3.2.4.2 Systemanalyse im z-Bereich

Als Analysemethode im z-Bereich wird primär die Pol-/Nullstellenverteilung von G(z) in der z-Ebene herangezogen. Der Befehl

pzmap(sys)

stellt diese Pol-/Nullstellenverteilung dar (o: Nullstelle, x: Polstelle). Mit dem nachfolgenden Befehl "**zgrid**" wird u.a. auch der Stabilitätsbereich in der z-Ebene, der Einheitskreis, eingezeichnet.

3.2.4.3 Systemanalyse im Frequenzbereich

Der Befehl

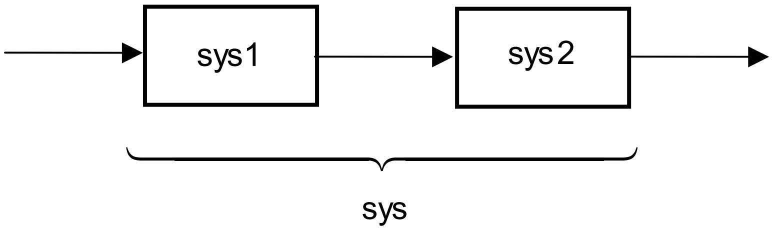
[betrag, phase] = **bode** (sys, w);

berechnet den Betrag und die Phase des Frequenzganges $G(j\omega_Z)$ sofern "sys" ein zeitdiskretes System beschreibt. "w" ist der Vektor des Abszissenbereichs von ω, über dem Betrag und Phase berechnet werden sollen. w wird zweckmäßig wieder mit dem Befehl **logspace** berechnet. Sonst ist das Programm, wie es bei der Berechnung von Bodediagrammen kontinuierlicher Systeme vorgestellt wurde, auch hier verwendbar.

3.3 Verknüpfung von Systemen

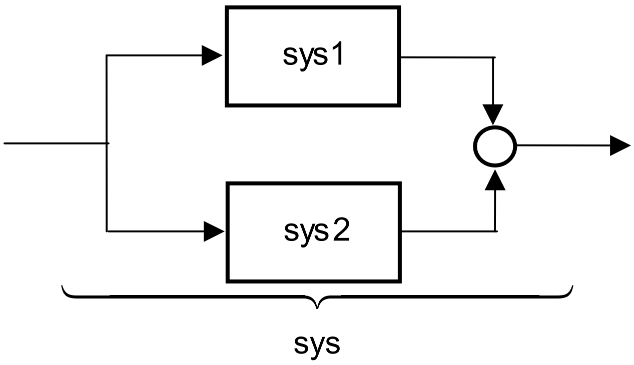
Zur Verknüpfung von Teil-Übertragungssystemen, die als LTI-Objekte (z.B. Übertragungsfunktions-Objekte) vorliegen, zu einem Gesamtsystem benutzt man die gleiche Notation, wie man sie zur Handrechnung benutzt:

- Berechnung der Gesamtübertragungsfunktion "sys" einer **Reihenschaltung** zweier Systeme "sys1" und "sys2":



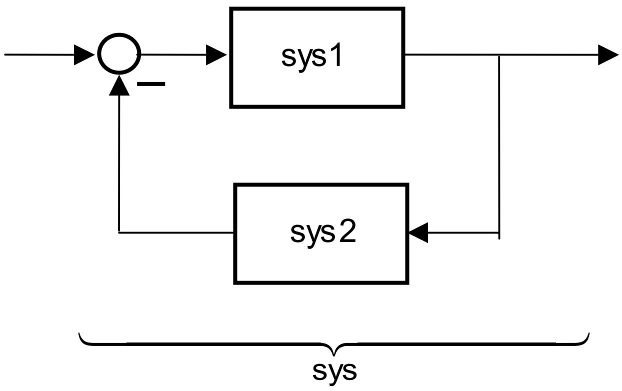
sys = sys1 * sys2;

- Berechnung der Gesamtübertragungsfunktion "sys" einer **Paralleleschaltung** zweier Systeme "sys1" und "sys2":



sys = sys1+ sys2;

- Berechnung der Gesamtübertragungsfunktion "sys" einer **Kreisschaltung** zweier Systeme "sys1" und "sys2":



sys = sys1 / (1 + sys1 * sys2);

3.4 Ein Demonstrationsprogramm zur Anwendung von Matlab in der Systemtheorie

Im folgenden wird ein Matlab-Programm angegeben, daß die Funktion nahezu aller in Kapitel 3.1 und 3.2 beschriebenen Befehle zur Systemtheorie in einem Script-File demonstriert.

```
% Programm "ml_demo_5_3".
% Demonstrationsprogramm zur Anwendung von Matlab 5.3
% in der Systemtheorie zur Analyse kontinuierlicher
% und zeitdiskreter Systeme im Zeit-, Bild- und
% Frequenzbereich.
%
% (Die als Berechnungs- oder Umrechnungsbefehle
% bezeichneten Matlab-Befehle führen im Kern die in
% den jeweiligen Programmteil-Überschriften genannten
% Berechnungen aus)

clear                    % Arbeitsspeicher säubern
home                     % Cursor in linke obere Bildecke
close all                % alle Grafiken schließen
format short e           % genauere Zahlendarstellung

% Eingabe des Systems in Form seines Zustandsmodells.
%
disp('Eingegebenes Zustandsmodell')
                         % Textausgabe
%A=[0,0,-0.2;1 ,0,-1.1;0,1,-0.7];
                         % Systemmatrix
%b=[5;10;0];             % Eingangsvektor
%c=[0,0,1];              % Ausgangsvektor
%d=0;                    % Durchgangsfaktor

% Alternativ wählbares System
%
A=[0,0,-6;1,0,-11;0,1,-6];
b=[1;0;0];
c=[0,0,1];
d=0;
%
k_ss_sys=ss(A,b,c,d)     % kontinuierliches Zustandsmodell-Objekt

pause                    % stoppt den Programmlauf bis
                         % CR-Eingabe.
home

% Übergang von Zeitbereich in den Bildbereich:
% Berechnuung der Übertragungsfunktion in Polynomform
% aus dem Zustandsmodell.
%
disp('Übertragungsfunktion in Polynomform')
[k_tf_sys]=tf(k_ss_sys)  % Wandlungsbefehl
pause
home
```

```
% Extrahierung des  Zähler- und Nennerpolynoms der
% Übertragungsfunktion aus dem Übertragungsfunktions-
% Objekt "k_tf_sys".
%
[zk_cell,nk_cell]=tfdata(k_tf_sys);
                           % Berechnung der  Cell-Arrays
                           % des Zähler- und Nennerpolynoms.
zk=zk_cell{1};             % Koeff.-Vektor des Zählers;
nk=nk_cell{1};             % Koef.-Vektor des Nenners;


% Umrechnungen der Übertragungsfunktion in ihre
% verschidenen Darstellungsformen
%
% Umrechnung der Übertragungsfunktion in Produktform
%
disp('Übertragungsfunktion in Produktform')
[null,pol,k]=tf2zp(zk,nk); % Umrechnungsbefehl
Zaehlernullstellen=null    % Ergebnisausgabe
Nennernullstellen=pol
Faktor=k
pause
home
%
% Umrechnung der Übertragungsfunktion in Partialbruchform
% (Berechnet keine sinnvollen Ergebnisse bei komplexen
% Pol- oder Nullstellen)
%
disp('Übertragungsfunktion in Partialbruchform')
[r,p,k]=residue(zk,nk);    % Umrechnungsbefehl
Zaehler=r                  % Ergebnisausgabe
Pole=p
Faktor=k
pause
home
%
% Umrechnung der Übertragungsfunktion in die V-Normalform
%
disp('Übertragungsfunktion in V-Normalform')
tf2vn(zk,nk);              % Umrechnungsbefehl,
                           % gehört nicht zum Matlab-
                           % befehlsvorrat.
pause
home

% Berechnung von Systemantworten
%
tend=25;                   % Darstellungsintervall-Länge
dt=0.01;                   % Rechenschrittweite
t=0:dt:tend;               % Zeitachse
figure(1)                  % erster Grafik-Bildschirm
%
% Impulsantwort
%
g=impulse(k_ss_sys,t);     % Berechnungsbefehl
subplot(3,1,1)             % Plot in erste "Zeile" von drei
                           % möglichen.
plot(t,g)                  % eigentlicher Plot-Befehl
grid on                    % Gitternetz über Grafik
title('Impulsantwort')     % Titel über Grafik
```

```
%
% Sprungantwort
%
h=step(k_ss_sys,t);         % Berechnungsbefehl
subplot(3,1,2)              % Plot in zweite "Zeile" von drei
                            % möglichen.
plot(t,h)
grid on
title('Sprungantwort')
%
% Antwort auf eine beliebige Erregung
%
u=stepfun(t,0)-stepfun(t,5);  % Erregungssignalgenerierung:
                              % hier: Puls von 5sek. Dauer,
                              % Ampliude=1
[y,t,x]=lsim(k_ss_sys,u,t);   % Berechnungsbefehl
subplot(3,1,3)                % Plot in dritte "Zeile" von drei
                              % möglichen.
plot(t,y,'y',t,u,'r')         % Ausgabe der Einganserregung (rot)
                              % und der Systemantwort (gelb).
grid on
title('Beliebige Erregung (rot eingezeichnet)')
xlabel('t / sek')             % Beschriftung der x-Achse
pause
home
%
% Zustandsgrößenverläufe bei der oben gewählten Erregung
%
figure(2)                     % zweiter Grafik-Bildschirm
subplot(3,1,1)
plot(t,x(:,1))                % 1. Spalte der Zustandsgrößen x
                              % (siehe "lsim")
grid on
title('Zustandsgröße 1 bei beliebiger Erregung')
subplot(3,1,2)
plot(t,x(:,2))                % 2. Spalte der Zustandsgrößen x
                              % (siehe "lsim")
grid on
title('Zustandsgröße 2 bei beliebiger Erregung')
subplot(3,1,3)
plot(t,x(:,3))                % 3. Spalte der Zustandsgrößen x
                              % (siehe "lsim")
grid on
title('Zustandsgröße 3 bei beliebiger Erregung')
xlabel('t / sek')
pause
home

% Systemanlayse im Bildbereich:
% Pol-/Nullstellenverteilung
figure(3)                     % dritter Grafikbildschirm
pzmap(k_ss_sys)               % Berechnungsbefehl
                              % mit direkter Grafikausgabe
grid
title('Pol- / Nulstellen-Plan')
pause

% Systemanalyse im Frequenzbereich:
% Bodediagram
w=logspace(-2,2,600);           % Omega-Achse
```

```
[betrag,phas]=bode(k_ss_sys,w);  % Berechnungsbefehl
bdB=20*log10(betrag);            % dB-Berechnung

bdB=reshape(bdB,[1,length(w)]);
phas=reshape(phas,[1,length(w)]);
                              % Wandlung der von "bode"
                              % erzeugten 3-D Matrizen in
                              % Zeilen-Vektoren (zum Plotten).

figure(4)                     % vierter Grafikbildschirm
subplot(2,1,1)                % Plot der Betragskennlinie
                              % in erste "Zeile" von
                              % zwei möglichen
semilogx(w,bdB)               % Plot-Befehl für logarith-
                              % mische x-Achse
grid on
title('Bodediagramm: Betragskennlinie')
ylabel('dB')                  % Beschriftung der y-Achse
subplot(2,1,2)                % Plot der Phasenkennlinie
                              % in die zweite "Zeile" von
                              % zwei möglichen
semilogx(w,phas)
grid on
title('Phasenkennlinie')
xlabel('omega [1/sek]')
ylabel ('Grad')

% Übergang zur Zeitdiskreten Systemen
%
% Diskretisierung des kontinuierlichen Systems
%
T=1;                             % gewählte Abtastzeit
disp('Zeitdiskrete Übertragungsfunktion (Polynomform)')
d_tf_sys=c2d(k_tf_sys,T,'zoh')   % Umrechnungsbefehl
pause
home

% Diskrete Systemantworten
%
td=0:T:tend-T;                   % diskrete Zeitachse

%
% Diskrete Impulsantwort
%
gd=impulse(d_tf_sys,td);         % Berechnungsbefehl
figure(5)
subplot(3,1,1)
stairs(td,gd)
grid on
title('Diskrete Impulsantwort')
%
% Diskrete Sprungantwort
%
gd=step(d_tf_sys,td);            % Berechnungsbefehl
subplot(3,1,2)
stairs(td,gd)
grid
title('Diskrete Sprungantwort')
%
```

```
% Diskrete Systemantwort auf beliebige Erregung
%
ud=stepfun(td,0)-stepfun(td,5);
                              % Erregungssignalgenerierung
                              % Puls von 5sek. Dauer,
                              % Ampliude=1.
yd=lsim(d_tf_sys,ud,td);      % Berechnungsbefehl
subplot(3,1,3)
stairs(td,yd,'y')
hold on
stairs(td,ud,'r')
grid on
hold off
title('Beliebige Erregung (rot eingezeichnet)')
xlabel('t / sek')
pause
home

% Systemanalyse im Bildbereich:
% Pol-/Nullstellenverteilung des diskreten Systems
%
figure(6)
pzmap(d_tf_sys)                 % Berechnungsbefehl
zgrid                           % spezielles Gitternetz
                                % für zeitdiskrete Systeme
title('Pol- / Nulstellen-Plan')
pause

% Systemanalyse im Frequenzbereich:
% Bodediagram des diskreten Systems
%
w=logspace(-2,2,800);
[betrag,phas]=bode(d_tf_sys,w); % Berechnungsbefehl
bdB=20*log10(betrag);

bdB=reshape(bdB,[1,length(w)]);
phas=reshape(phas,[1,length(w)]);
                                % Wandlung der von "bode"
                                % erzeugten 3-D Matrizen in
                                % Zeilen-Vektoren (zum Plotten).
figure(7)
subplot(2,1,1)
semilogx(w,bdB)
grid on
title('Bodediagramm: Betragskennlinie')
ylabel('dB')
subplot(2,1,2)
semilogx(w,phas)
grid on
title('Phasenkennlinie')
xlabel('omega [1/sek]')
ylabel ('Grad')
```

3.5 Regelungstechnik

Die meisten für die klassische Regelungstechnik notwendigen Matlab-Befehle sind schon vorangehend im Rahmen der Befehle für die Systemtheorie aufgeführt. Da wir uns auf diese beschränken wollen (d.h. wir betrachten keine Matlab-Befehle für die Regelungstechnik im Zustandsraum), verbleiben nur noch ein spezieller Befehl für Bodediagramme von offenen Regelkreisen und zwei Befehle zum Wurzelortsverfahren:

3.5.1 Phasenrand und Durchtrittsfrequenz

Der Matlab-Befehl

[A_r, Phi_r, omega_Ar, omega_c] = **margin** (sys);

berechnet aus Übertragungsfunktions-Objekt "sys"

den Amplitudenrand A_r *):	A_r,
den Phasenrand Φ_r (in Grad):	Phi_r,
die Frequenz ω_{-180} bei der der Amplitudenrand auftritt:	omega_Ar,
die Durchtrittsfrquenz ω_c:	omega_c.

*)Um den Amplitudenrand in Dezibel (dB) zu erhalten muß A_r noch der Operation A_rdB= 20*log10(A_r) unterzogen werden.

Der Befehl liefert natürlich nur dann sinnvolle Ergebnisse, wenn das Übertragungs-funktions-Objekt "sys" aus der *Übertragungsfunktion eines offenen Regelkreises L(s)* hervorgegangen ist.

Wird er Befehl ohne linksseitige Argumente aufgerufen:

margin (sys);

wird das Bodediagramm des offenen Kreises mit eingezeichnetem Amplituden- und Phasenrand und den gekennzeichneten, dazugehörigen Frequenzen grafisch dargestellt. Als Bildüberschrift werden diese Werte numerisch ausgegeben. Dabei bedeuten:

Gm: (Gain margin) Amplitudenrand in dB, dahinter in Klammern die dazugehörige Auftrittsfrequenz ω_{-180}

Pm: (Phase margin) Phasenrand in Grad, dahinter in Klammern die Durchtrittsfrequenz ω_c

3.5.2 Die Wurzelortskurve

Die Befehlsfolge

sys = **tf** (zL, nL);
rlocus (sys);

mit

zL: Zähler der Übertragungsfunktion des offenen Regelkreises in V-Normalform mit V-Regler = 1.
nL: Nenner der Übertragungsfunktion des offenen Regelkreises in V-Normalform mit V-Regler = 1.

zeichnet die Wurzelortskurve "WOK" (Root-Locus) , d.h. die Nullstellenlage des Nennerpolynoms eines geschlossenen Regelkreises

$$N_L(s) + K \cdot Z_L(s) = 0$$

in Abhängigkeit des Faktors "K" (welcher unter den gegebenen Randbedingungen der Verstärkungsfaktor des Reglers ist). Die K-Variation wird automatisch vorgenommen, so daß ein glatter verlauf der WOK entsteht.

Um feststellen zu können, welcher Faktor K für einen bestimmten Punkt der WOK gilt, wird der Befehl

[K, pole] = **rlocfind** (sys)

im Anschluß an **rlocus** benutzt. Der Befehl setzt einen Haarkreuz-Cursor auf die Grafik, der mit der Maus auf eine beliebige Stelle der WOK geschoben werden kann. Mit einem Mausklick werden dann der dazugehörige Wert von K und die angeklickte Polstelle des geschlossenen Regelkreise im MCW ausgegeben. Auf der WOK werden alle Polstellen mit einem "+"-Zeichen markiert, die zu diesem K-Faktor gehören. Für weitere K-Analysen muß der Befehl immer neu aufgerufen werden.

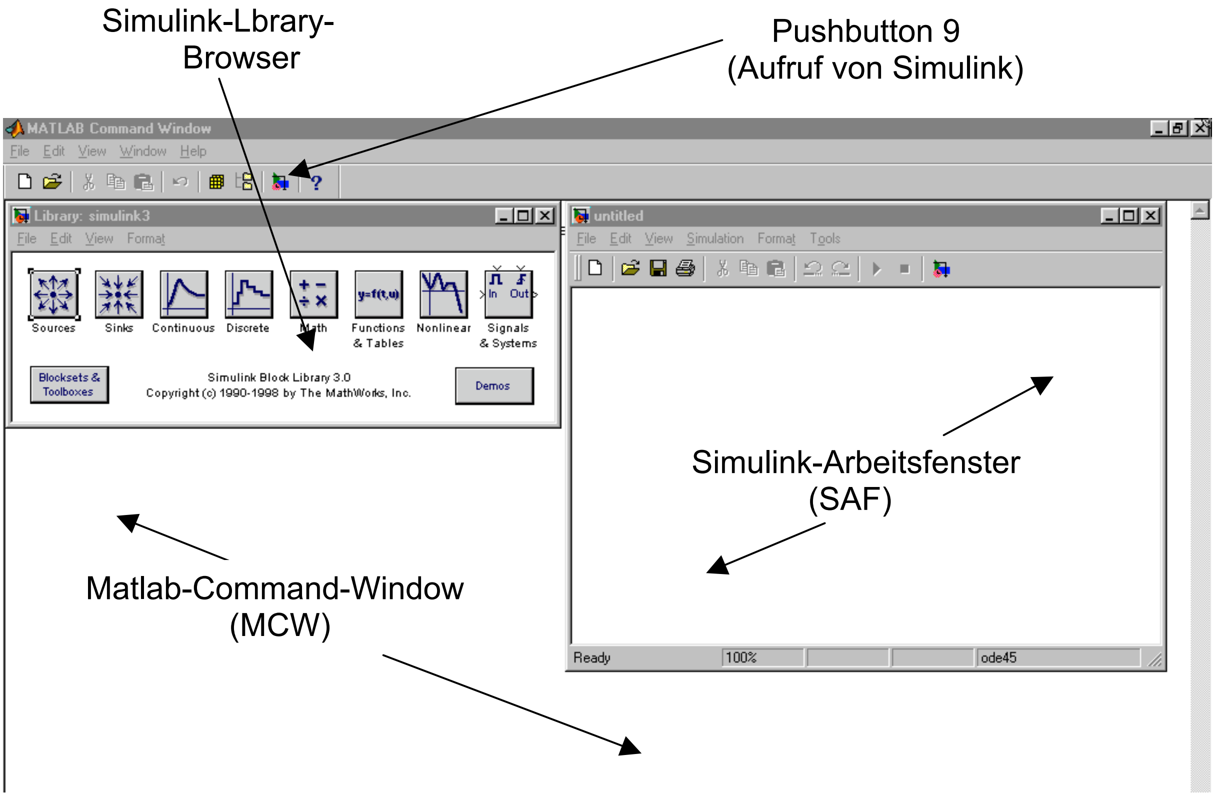
4 Einführung in Simulink

Simulink ist eine Matlab-Toolbox zur grafikunterstützten Eingabe und Zeitbereichs-Simulation weitgehend beliebig vermaschter (auch nichtlinearer) Modellstrukturen von Übertragungssystemen.
Die zu simulierenden Modelle werden dabei aus einzelnen grafischen Blöcken, die in Bibliotheken ("Block-Librarys") zur Verfügung gestellt werden, aufgebaut. Die Modelle können in Form von Grundelementen (Integrierer, Summierer, usw.) oder beliebig dimensionalen Zustandsmodellen und auch als Übertragungsfunktionen eingegeben werden. Auch Mischungen aus Bild- und Zeitbereichs- oder kontinuier-lichen und zeitdiskreten Systemelementen sind erlaubt.
Der Vorrat an Simulationsblöcken umfaßt lineare, nichtlineare und zeitdiskrete Elemente, mit denen auch Mehrgrößen-, nichtlineare und zeitvariable Systeme nachgebildet werden können.
Durch grafikunterstützte Hinzufügung von auswählbaren Erregungsquellen und Signalanzeige-Elementen (z.B. "Oszilloskope") lassen sich Simulationen im Zeitbereich unter Simulink ohne Kenntnisse der Syntax von Matlab durchführen. Durch Schnittstellen zwischen Simulink und Matlab besteht die Möglichkeit, unter Simulink erzeugte Modellstrukturen auch unter Matlab weitergehend zu analysieren, z.B. durch die Berechnung von Bodediagrammen oder Wurzelortskurven. Auch die Übergabe unter Matlab berechneter Parameter und Programmstrukturen in Simulink-Simulationsstrukturen sind möglich.
Simulink kann durch eine Reihe von sogenannten Blocksets in seiner Funktionalität wesentlich erweitert werden. So stellt z.B. das sogenannte DSP-Blockset (Digital-Signal-Processing-Blockset) neben Blöcken zur Erzeugung digitaler Filter auch Simualtionselemente zur Spektralanalyse von Signalen zur Verfügung.
Das wesentlichste Erweiterungs-Blockset von Simulink ist der sogenannte Real-Time-Workshop, der es ermöglicht, mit Simulink-Strukturen über geeignete Hardware mit A/D-D/A-Wandlern in Echtzeit auf die reale Umwelt zuzugreifen /6/. Damit sind Methoden des "Rapid Prototyping", d.h. des schnellen Entwurfs und der schnellen Erprobung von Filter und Regelalgorithmen in realen Arbeitsumgebungen möglich.
Wir konzentrieren uns in dieser Einführung auf das Grundsystem Simulink ohne Erweiterungs-Blocksets. aus Zeitgründen können wir in dieser Einführung wiederum nur einen Bruchteil der Leistungsfähigkeit von Simulink darstellen.

4.1 Grundlagen der Bedienung von Simulink

Nach Eingabe des Zeichenfolge "**simulink**" (CR) im MCW oder Anklicken des Simulink-Pushbuttons (**9**) im MCW öffnet sich der ***Simulink Library Browswer*** mit mit dem auf die Simulink Simulationselemente zugegriffen werden kann. Wir benutzen einen anderen Aufruf, nämlich ***simulink3*** (CR). Bei Eingabe dieses Aufrufs wird ein grafisches Menü der Block-Library angezeigt (siehe das folgende Bild).

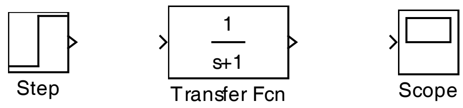
Zu Beginn einer neuen Simulink-Sitzung klickt man im Pulldown-Menü ***File*** dieses Library-Fensters die Zeile ***New -> Model*** an und es öffnet sich ein weiteres Fenster mit dem Namen Untitled. Dies ist die grafische Arbeitsebene von Simulink, die wir ***Simulink-Arbeits-Fenster (SAF)*** nennen wollen. Damit sind alle Voraussetzungen zum Beginn einer Simulink-Sitzung gegeben.



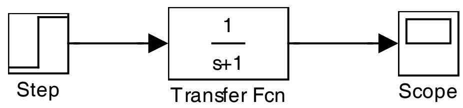
Die Simulink Block-Library und das Simulink Arbeitsfenster (SAF)

Hinter den Icons ***Sources, Sinks, ..., Signals & Systems*** des Library-Fensters verbergen sich die Simulationselemente von Simulink. Durch einen Doppelklick auf ein solches Icon öffnet sich ein weiteres Fenster und zeigt die Simulations-Elemente der gewählten Kategorie an. Mittels "click + drag" lassen sich Kopien dieser Simulationselemente in das SAF ("Untitled") ziehen.

Zu unserer ersten Simulink-Sitzung holen wir uns aus der ***Sources***-Bibliothek den Block ***Step*** (Eingangs-Sprungfunktion), schließen diese Bibliothek und öffnen die ***Continuous***-Bibliothek, um uns dort den Block ***Transfer Fcn*** (Transfer Function, Übertragungsfunktion) in das SAF zu ziehen.
Nach Schließung dieser Bibliothek öffnen wir schließlich die ***Sinks***-Bibliothek und kopieren uns den Block ***Scope*** (Oszilloskop) in die grafische Arbeitsebene und ordnen die Blöcke wie folgt an:



Durch "click + drag" können wir vom Ausgang des ***Step**-Blocks* zum Eingang der ***Transfer Fcn*** eine Verbindung ziehen. Lassen wir die Maustaste los, entsteht ein Flußrichtungspfeil zwischen den beiden Objekten:



Mit den gleichen Vorgehensschritten läßt sich dieser Pfeil auch zwischen ***Transfer Fcn*** und ***Scope*** herstellen.

Zur Parametrierung dieser Simulationsblöcke müssen diese doppelt angeklickt werden. Es öffnet sich dann ein entsprechendes Fenster, das einige Informationen über das Simulationselement enthält und in das die gewünschten Parameterwerte eingetragen werden können.
Den Sprunggenerator ***Step*** parametrieren wir so, daß der Sprung bei t = 0 beginnt (***Step time*** = 0), der Sprunganfangswert (***Initial value***) = 0 und der Sprungendwert (***Final value***) = 1 sind, also ein Einheitssprung generiert wird. Mit einem ***OK*** -Klick werden diese Parameter übernommen und das Fenster geschlossen.

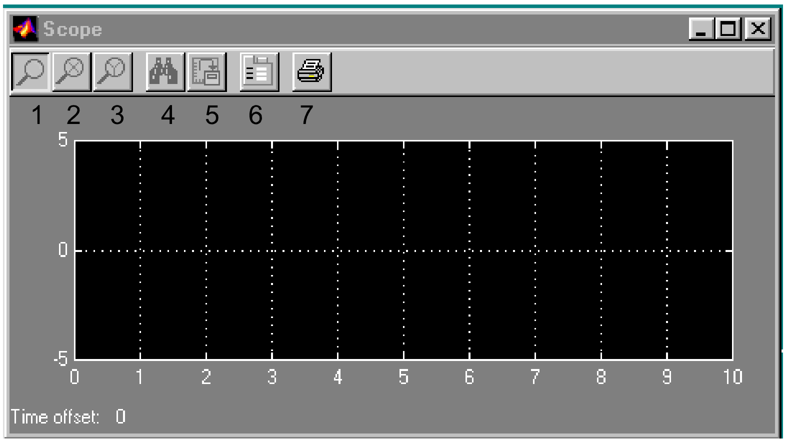
Durch Anklicken von **Help** öffnet das helpdesk für eine nähere Erläuterung der Funktionsweise und der Parametrierung des aktuellen Blocks.

Als Systemmodell wollen wir ein gedämpft schwingfähiges PT_2-Glied simulieren

$$G(s) = \frac{3{,}7}{1 + 2 \cdot 0{,}5 \cdot 2{,}5s + (2{,}5s)^2}\,.$$

In den Parametrierungsblock der ***Tansfer Fcn*** muß dementsprechend als ***Numerator*** (Zähler) = [3.7] und als ***Denominator*** (Nenner) = [6.25, 2.5, 1] eingetragen werden.

Nach Schließung dieses Fensters durch das Anklicken von ***OK*** wird der ***Scope***-Block angeklickt. Es öffnet sich ein Grafik-Fenster mit dem Aussehen eines
Oszilloskops, über dem sich sieben Buttons mit folgender Bedeutung befinden (siehe folgendes Bild).

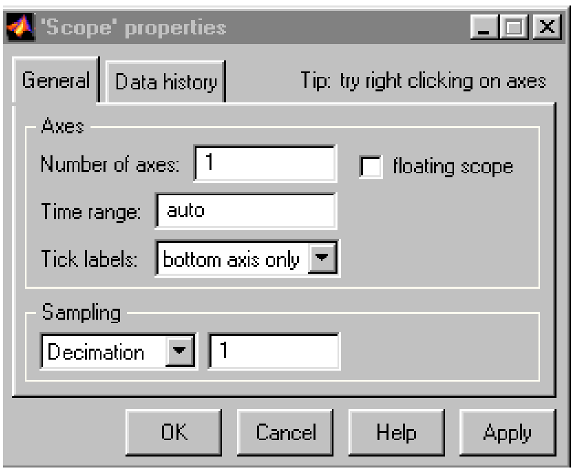


1: Zoom in x- und y-Richtung
2: Zoom in x-Richtung
3: Zoom in y-Richtung
4: Automatische Skalierung
5: Rettet die aktuellen Einstellungen
6: Eigenschaften einstellen
7: Graphen drucken

Das Simulink-Scopefenster

Die Bedeutung der Buttons ist weitgehend selbsterklärend: mit ***1***, ***2*** und ***3*** kann mit Hilfe der Maus nach Betätigung des entsprechenden Buttons und anschließendem 'klicken und ziehen' der Bildinhalt entsprechend gezoomt werden. Button ***4*** nimmt durch Anklicken eine automatische Skalierung vor, so daß der gesamte Bildinhalt, wie bei Matlab gewohnt, optimal in den Koordinaten angezeigt wird. Mit Button ***5*** kann eine als optimal erkannte Skalierung für die folgenden Signalausgaben "eingefroren" werden.

Der zunächst wichtigste Button ist Nr. ***6 "Eigenschaften einstellen".*** Betätigt man ihn, öffnet sich ein Fenster ***'Scope' properties*** mit zwei weiteren Ordnern ***General*** und ***Data history***.



Der Ordner General im Gafik-Eigenschaften-Fenster

Im Ordner ***General*** werden folgende Eigenschaften abgefragt:

- ***Number of axes***
  Trage Sie hier ein, wieviel Darstellungs-Kanäle der Scope haben soll. So erhält man z.B. bei einem Eintrag von z.B. "2" zwei Scope-Eingänge und zwei übereinander liegende Bildschirme. Es wird empfohlen mit nur einem Eingang pro Scope und dann mit mehreren Scopes zu arbeiten.

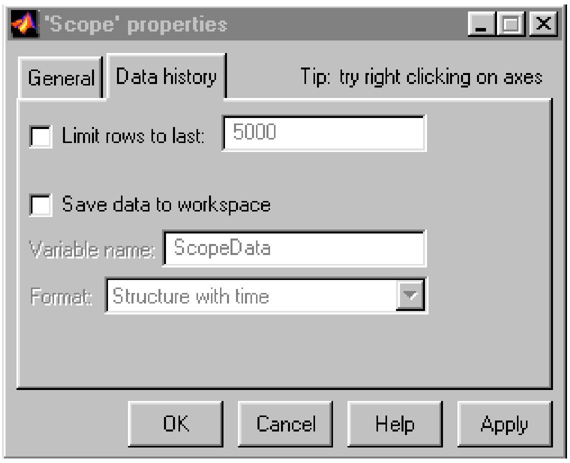
- ***Time range***
  Hier wird eingetragen bis zu welchem Endzeitpunkt die Signale im Scope ausgegeben werden sollen. Bei ***auto*** ist die Zeitachse genau so lang, wie die bei der Parametrierung im ***SAF-Pulldown-Menü Simulation -> Parameters -> Solver*** eingegebene ***Stop Time*** (Wir kommen gleich auf diese wichtigen Parameter-Einstellungen zurück). Hier kann aber auch jede Zeit, die kleiner ist als die ***Stop Time,*** eingegeben werden, um eine repetierende, aber besser aufgelöste Signaldarstellung zu erzielen.

- ***Tick labels***
  Bottom axis only beschriftet bei der Wahl von ***Number of axes*** > 2 nur die untere Zeitachse, ***all*** alle Graphen und ***non*** keine.

- ***Sampling***
  Im Feld ***Sampling*** kann in der Auswahl ***Sampling time*** eine <u>Darstellungs-Abtastzeit</u> für die Scope-Darstellung des Signals gewählt werden. Bei Nutzung der Auswahl ***Decimation*** kann die Anzahl der Darstellungs-Stützpunkte dezimiert werden. Wenn z.B. die Simulation mit einer Auflösung von dt = 0,1 sek. durchgeführt und Decimation=10 gewählt wird, ergibt sich eine Darstellungs-Abtastzeit von 1 sek.

Im zweiten Ordner des Fensters ***'Scope' properties*** (siehe folgendes Bild) nämlich in ***Data history*** deaktivieren wir (falls es nicht schon geschehen ist), ***Limit rows to last ...*** . Damit stellen wir sicher, daß bei der Scope - Anzeige keine Darstellungsverluste durch die hier einstellbare Maximalanzahl von Bildpunkten entstehen.

Die nächste Auswahl ***Save data to workspace*** ist für Echtzeitanwendungen /6/ und für <u>Dokumentationszwecke</u> wichtig.
Will man einen unter Simulink auf einen Scope-Block erzeugten Graphen in ein Textverarbeitungssystem (z.B. Word) bringen, muß man, um mit Vektor-Grafiken arbeiten zu können, einen Umweg über Matlab-Plots machen. Dieser Vorgang soll hier beschreiben werden.



Der OrdnerData History im Grafik-Eigenschaften-Fenster

Wir aktivieren dazu das Feld ***Save data to workspace*** und wählen als ***Format*** "***Matrix (compatible with V2.0-2.2)".*** Die Scope-Daten werden dann in einen Vektorformat, daß sich leicht weiterverarbeiten läßt, im Matlab-Arbeitspeicher unter dem Namen abgelegt, den Sie unter ***Variable name*** eintragen. Im vorliegenden Fall ist es der Default-Eintrag ***ScopeData***.

Nach Ablauf der Simulationsdauer stehen Ihnen nun die Grafikdaten unter Matlab zur Verfügung. Mittels ***whos*** erfolgt im Matlab-Command-Window folgende Ausgabe:

```
Name             Size              Bytes   Class
ScopeData        51x2               816    double array
Grand total is 153 elements using 1224 bytes
```

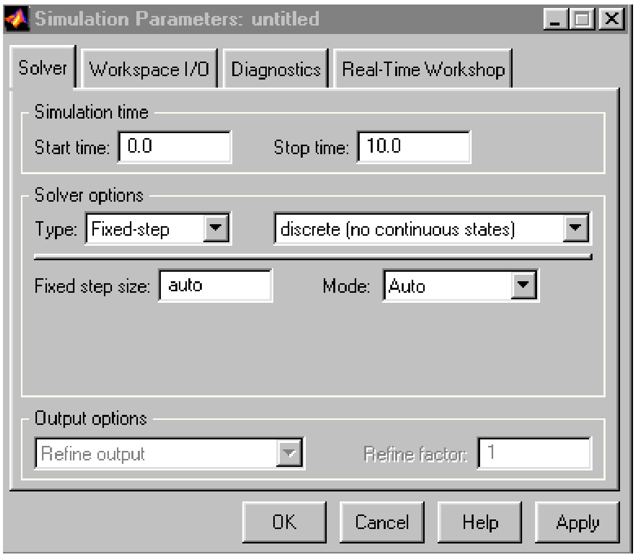
In der (n x 2) - Matrix "ScopeData" befinden sich in der ersten Spalte die Zeitachse "ScopeData(: , 1)" und in der zweiten Spalte der Signalverlauf "ScopeData(: , 2)".
Der Signalverlauf könnte z.B. mit dem Befehl

```
plot(ScopeData(:,1), ScopeData(:,2)), grid on
```

unter Matlab geplottet werden. Die abschließende Übergabe dieser Matlab-Grafik in ein Textverarbeitungssystem wurde im Kapitel 2.6.2 beschrieben.

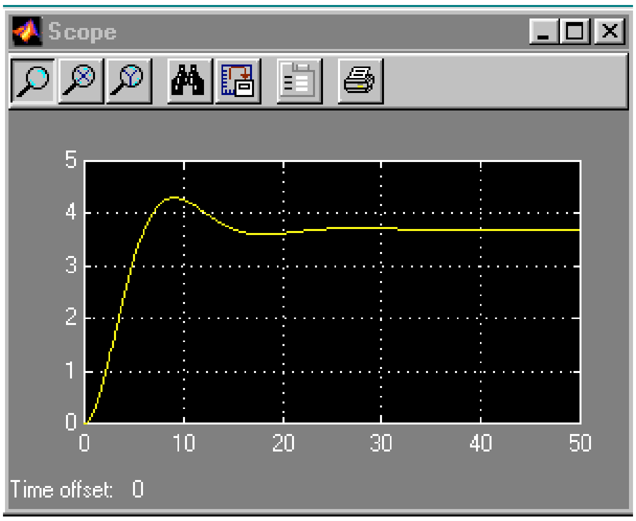
Für unsere erste Simulink-Anwendung klicken wir auf den ***Scope-Button 6*** und setzen die ***Scope-properties*** im Ordner ***General*** wie folgt: ***Number of axes***: 1; ***Time range***: auto; und belassen die restlichen Default-Einstellungen.
Im Ordner ***Data history*** deaktivieren wir, wie oben erläutert, ***Limt rows to last ...*** und lassen ***Save data to workspace*** unaktiviert.

Zum Starten der Simulation wählen wir im SAF-Pulldown-***Menü Simulation*** zunächst den ***Menü***punkt ***Parameters***. In dem sich öffnenden Fenster interessiert uns nur der Ordner ***Solver***.



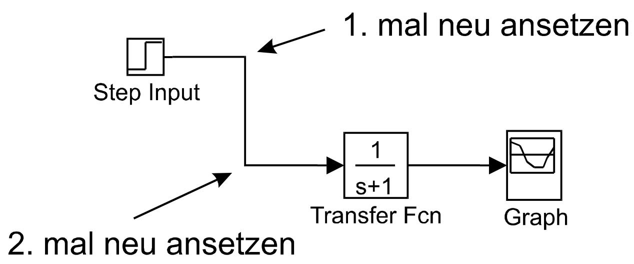
Der Ordner Sover im Simulation-Menü des SAF

Im Abschnitt ***Simulation time*** werden die ***Start time*** der Simulation (i.a. 0.0) und die ***Stop time***, also die Simulationsdauer, beide in Sekunden eingegeben. Für unser Beispiel wählen wir eine ***Stop time*** = 50. Unter ***Solver options*** wählen wir ***Fixed step***, und ***ode5(Runge-Kutta)*** sowie eine ***Fixed step size*** von 0.1 ein. Das Fenster ***Simulation Parameters*** kann nun geschlossen werden (***OK***).
(Nähere Einzelheiten zur Bedeutung der vorgenommen Eintragungen werden später bei der Beschreibung aller Pulldown-Menüs des SAF beschrieben).
Anschließend wird im SAF ***Simulation***-Menü der Menüpunkt ***Start*** gewählt. Der Simulationsvorgang startet dann, was an der sich aufbauenden Sprungantwort im Scope-Fenster erkennbar ist:

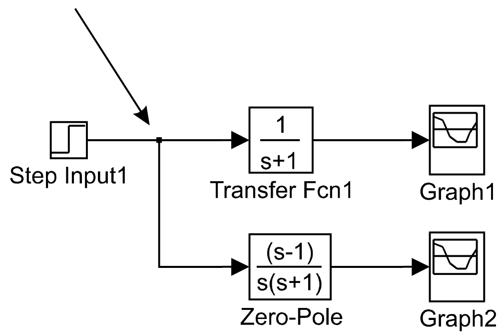


Nach diesem kurzen Beispiel wollen wir uns etwas intensiver mit den weiteren Eigenschaften von Simulink auseinandersetzen.

Weiter vorn haben wir beschrieben, daß mit "click + drag" (linke Maustaste) eine horizontale Verbindung zwischen einem Block-Ausgang und einem Block-Eingang hergestellt werden kann. In gleicher Weise, jeweils durch neues Ansetzen des Mauszeigers, können auch rechtwinklige Verbindungen erzeugt werden:



Eine Signalverzweigung der Form



wird hergestellt, indem die gewünschte Verzweigungslinie mit der rechten Maustaste im "click + drag"-Verfahren aus dem Verzweigungspunkt herausgezogen wird.

Durch einfaches Anklicken der Simulationsblöcke oder der Flußrichtungspfeile im SAF können diese markiert werden. In diesem Zustand können sie gelöscht (z.B. Taste ***Entf***) oder in einer anderen Weise manipuliert werden. So ist es z.B. möglich, sie durch "click + drag" zu verschieben oder die Größe des Blockes zu verändern. Dies geschieht durch "click“ auf einen Eckpunkt (dieser Zustand wird durch einen geneigten Doppelpfeil gekennzeichnet) und Bewegung der Maus. Wiederum mit "click + drag" (es erscheint ein Fadenkreuz) kann ein Kasten um die Simulationsstruktur (oder auch Teile davon) gezogen werden. Nach Loslassen der Maustaste sind alle Komponenten innerhalb des Kastens markiert.
Wiederum durch einfaches oder mehrfaches Anklicken läßt sich der Name unter jedem Simulationsblock markieren (graue Unterlegung). In diesem Zustand kann der bestehende Name durch einen anderen gewünschten ersetzt werden (Tastatur).

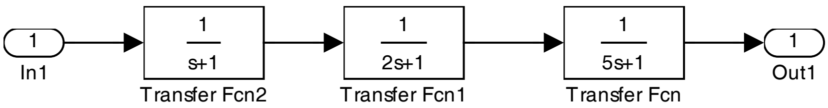
Die **<u>Menüpunkte des Simulink-Arbeitsfensters (SAF)</u>** leisten folgendes (Es werden nur die wichtigsten Menüpunkte beschrieben):

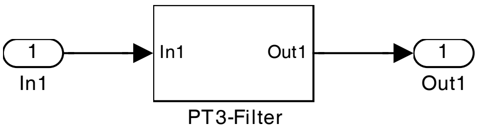
- **SAF File-Menü**
  - ***New***: Öffnet eine neue grafische Arbeitsebene.
  - ***Open***: Lädt eine bereits gespeicherte Simulationsstruktur aus vorangegangenen Sitzungen in die grafische Arbeitsebene.
  - ***Close***: Schließt die grafische Arbeitsebene.
  - ***Save***: Speichert eine Simulationsstruktur (einschließlich aller Umgebungsparameter) unter ihrem aktuellen Namen (im Beispiel "Untitled") auf einem externen Speichermedium.
  - ***Save As***: Speichert eine Simulationsstruktur unter einem frei zu wählenden Namen und fügt automatisch die Namensergänzung ***.mdl*** (Model) an. Damit sind Simulink-Simualtionsstrukturen von anderen Matlab-programmen (**.m**, **.mat**) unterscheidbar.
  - ***Print***: Druckt die aktuell in der Arbeitsebene befindliche Simulationsstruktur.

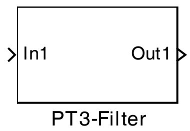
- **SAF Edit-Menü**
  - ***Cut, Copy, Paste***: Ausschneiden , Kopieren und Einfügen von markierten Simulationsstrukturen oder Teilelementen in eine Simulink-spezifische Zwischenablage, auf die von anderen Windows-Programmen nicht zugegriffen werden kann.
  - ***Select all***: Markierung der gesammten im SAF befindlichen Simulink-Struktur.
  - ***Copy Modell***: Kopiert die im SAF befindliche Simulink-Struktur in die Windows Zwischenablage.
  - ***Create Subsystem***: Erlaubt die Zusammenfassung einer umfangreichen Simulations-Struktur zu einem Symbol-Block. Signaleingänge und -Ausgänge in/aus diesen/m Block müssen dabei durch Inports ***In1*** bzw. Outports ***Out1*** (die sich in der später noch zu besprechenden *<u>Block-Library</u>* *<u>Signal & Systems</u>* befinden) dargestellt werden. Die zu gruppierende Struktur, einschließlich der Inports und Outports, <u>muß</u> mittels eines "click + drag - Kastens" gekennzeichnet werden.

    Beispiel: Zusammenfassung von drei in Reihe geschalteten PT1-Gliedern zu einem PT3-Filter (siehe folgende Grafik).

    Im ersten Schritt wird zusammenzufassende Struktur mit In- und Outports versehen. Nach der Zusammenfassung ergibt sich das mittlere Bild und nach Entfernung der nun nicht mehr notwendigen In- und Outports das letzte Bild, welches das Subsystem mit der obigen Simulationsstruktur enthält.







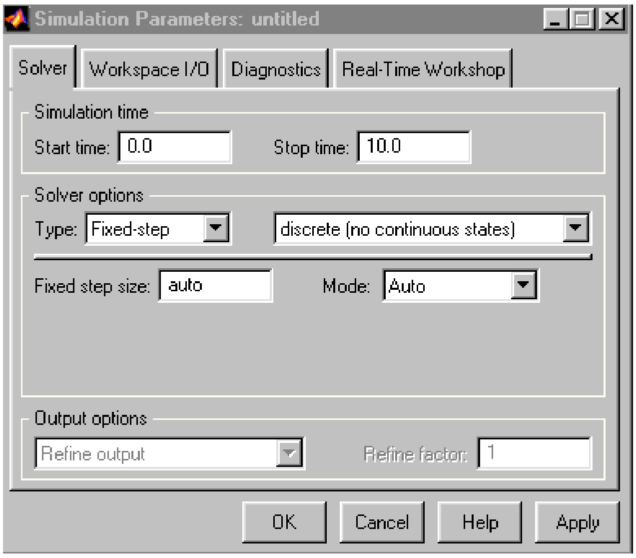
Die Anfügung von In- ind Outports hat den Sinn, bei mehreren Ein- und Ausgängen diese im Subsystem wieder Identifizieren zu können. Nach einem Doppelklick auf den Subsystem-Block öffnet sich dieser und die im Block enthaltenen Simulink-Struktur wird angezeigt.

- **SAF View-Menü**

  Verändert die Menüleisten des SAF und verändert Darstellungs-Parameter.

- **SAF Simulation-Menü**

  Diese Menü dient zum Einstellen aller wichtigen Simulationsparameter und zum Starten bzw. Stoppen einer Simulation.
  Falls der sogenannte Realtime-Workshop installiert ist, ist darauf zu achten, daß in diesem Menü der Menüpunkt ***Normal*** aktiviert ist. ***External*** muß aktiviert werden, wenn Simulink mit dem Realtime-Workshop in Echtzeit arbeiten soll. (Dieser Menüpunkt taucht taucht nicht auf, wenn der Realtime-Workshop nicht installiert ist.)
  Vor dem Starten einer Simulation wählen wir im Pulldown-Menü anschließend den Menüpunkt ***Parameters***. In dem sich öffnenden Fenster (siehe das folgende Bild) interessiert zunächst nur der Ordner ***Solver***.
  Im Abschnitt ***Simulation time*** wird die ***Start time*** der Simulation (i.a. 0.0) und die ***Stop time***, die Simulationsdauer, (beide in Sekunden) eingegeben. Die zu wählende ***Stop time*** hängt von der durchzuführenden Simulationsaufgabe ab. Sie kann beliebig groß gewählt werden (z.B. mit ***inf*** unendlich lang).
  Bei ***Solver-Options*** können in der Rubrik ***Type*** der Modus ***Fixed-step*** sowie ***Variable step*** und daneben ***discrete (no continous states)*** und verschiedene Integrationsverfahren (***ode x(...)***) für kontinuierliche Systeme gewählt werden.



Der Ordner Solver im Simulation-Menü des SAF

Wir arbeiten in dieser Einführung ausschließlich mit dem Modus ***Fixed-step***, also einer festen Rechen- (Integrations-) Schrittweite. Diese Schrittweite muß in das Feld ***Fixed step size*** eingetragen werden. Falls dort ***auto*** zu lesen ist, muß dieses ***auto*** mit dem numerischen Wert der noch zu bestimmenden Rechenschrittweite überschrieben werden. Im Feld ***Mode*** wählen wir in dieser Einführung immer ***Single Tasking***.

- **Simulation reiner kontinuierlicher Systeme**

Bei der Simulation kontinuierlicher Systeme sollte die Rechenschrittweite (***Fixed step size***) nach den Vorschriften des Kapitel 3.1.3.1 festgelegt werden.
Weiterhin muß eines der Integrationsverfahren (***ode x(...)***) z.B. ***ode 4(Runge-Kutta)*** ausgewählt werden. Bei unseren überwiegend linearen System ist es relativ unerheblich, welches Verfahren gewählt wird (auf das Integrationsverfahren ***ode1(Euler)*** sollte allerdings wegen seiner relativen Ungenauigkeit verzichtet werden).
Bei der Simulation nichtlinearer Systeme ist die Wahl häufig von Bedeutung und kann im helpdesk (Button ***help*** betätigen) und in /4/ nachgelesen werden.

- **Simulation reiner zeitdiskreter Systeme**

Bei der Simulation zeitdiskreter Systeme sollte als Rechenschrittweite (***Fixed step size***) die Abtastzeit T des zu simulierenden zeitdiskreten Systems benutzt werden. Rechts daneben muß der Modus ***discrete (no continous states)*** eingestellt werden.

- **Simulation gemischter kontinuierlicher und zeitdiskreter Systeme**

Bei Systemen, die aus kontinuierlichen und zeitdiskreten Simulationselementen bestehen, (wie so etwas realisiert wird, wird später erläutert) muß ein kontinuierlicher Integrationsalgorithmus (***ode x(...)***) benutzt werden, die ***Fixed Step size*** muß an den Erfordernissen der kontinuierlichen Systemelemente orientieren (also nach den Vorschriften des Kapitel 3.1.3.1 bemessen werden). Die Abtastzeit der zeitdiskreten Simulationselemente wird, wie wir in Kapitel 4.2 sehen werden, direkt in die zeitdiskreten Simulationblöcke eingetragen.

Liegt einmal eine lauffähige Version einer Simulation vor, kann auch ***Variable step size*** gewählt werden. Dies ist eine von Simulink gewählte variierende Schrittweite, die die Simulation wesentlich beschleinigen kann. Nähere Einzelheiten können im helpdesk (Button ***help*** betätigen) und in /4/ nachgelesen werden.
Das Fenster ***Simulation Parameters*** kann dann geschlossen werden (***OK***).

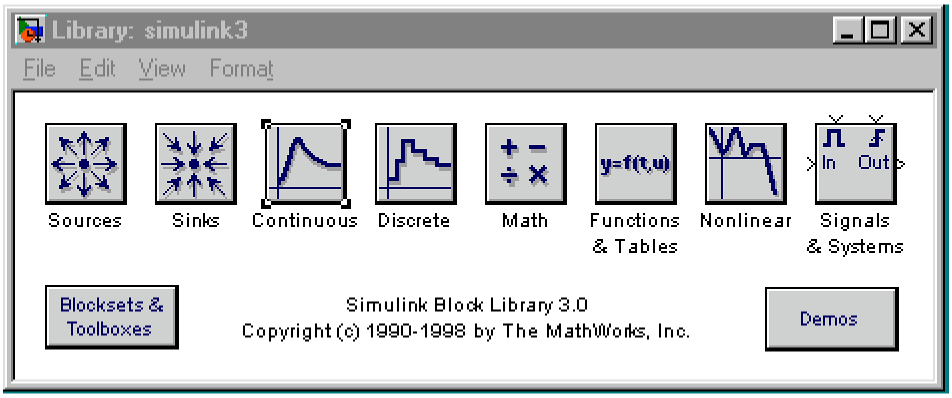
Zum Starten einer Simulation kann im SAF ***Simulation***-Menü der Menüpunkt ***Start*** oder im SAF der Start-Button (10. Button von links im SAF mit dem Pfeilsymbol) gewählt werden.

- **SAF Format-Menü**
  - ***Font***.Verändert das Beschriftungsformat der Simulink-Blöcke.
  - ***Flip/Hide Name***: Ändert die Beschriftungslage der Simulink-Blöcke bzw. entfernt sie.
  - ***Flip Block***: Dreht einen Simulink-Block pro Betätigung um jeweils 90°. Dient als grafische Konstruktionshilfe, wenn z.B. einem Block Signalleitungen von oben oder unten zugeführt werden sollen.
  - ***Rotate Block***: Dreht einen Simulink-Block um 180°. Dient als grafische Konstruktionshilfe z.B. bei Rückführungen.

- **SAF Tools-Menü**
  Wird in dieser einführung nicht benötigt.

4.2 Simulationselemente von Simulink

Nachdem wir im letzten Abschnitt die grundsätzlichen Funktionen von Simulink kennengelernt haben, wollen wir uns in diesem Kapitel mit den wichtigsten Simulationsblöcken der Simulink Block ***Library*** vertraut machen. Wir haben das äußere dieser Block-Library schon bei Öffnen von Simulink kennengelernt:



Die Simulink Block-Library

Nähere Informationen zur weitergehenden, hier nicht beschriebenen Parametrierungen und auch zu den hier nicht beschriebenen Blöcken, können wie folgt abgerufen werden:

1. Öffnen einer Block-Library: Es werden alle Simulationsblöcke der entsprechenden Library dargestellt.
2. Doppelklick auf einen Block: Es öffnet sich das Fenster zur Parametrierung des Blocks.
3. Anklicken des ***Help***-Buttons:

Es öffnet sich der helpdesk mit näheren Erläuterungen zu diesem Element. Falls diese Information nicht ausreicht, muß auf das Simulink-Handbuch /4/ zurückgegriffen werden.

In einigen Blöcken der Simulink-Block-Library wird eine ***Sample time*** abgefragt, die "***0***" gesetzt werden muß, falls der Block kontinuierlich arbeiten soll. Sonst muß hier die ***Abtastzeit T*** des zu simulierenden zeitdiskreten Systems eingesetzt werden. Bei Signalquellen sollte immer eine Eintragung nach den vorangehenden Erläuterungen erfolgen. Bei später folgenden Blöcken kann eine "***-1***" eingetragen werden, dann wird die ***Sample time*** - Parametrierung des Vorgänger-Blockes übernommen. (Wenn man an dieser Stelle einen Fehler macht, wird man von Simulink beim Starten einer Anwendung darauf aufmerksam gemacht.)

Von den folgenden einzelnen Librarys werden wieder nur die wichtigsten Simulationselemente erläutert.

- **Sources – Library**

Die Sources-Library stellt <u>Signalquellen</u> zur Erregung von Simulationsmodellen zur Verfügung:

Constant: Erzeugt einen konstanten Signalpegel oder einen konstanten Wert.

Signal Generator: Erzeugt Sinus-, Rechteck- und Sägezahnschwingungen sowie Rauschsignale. Die Frequenz kann in Hz oder rad/sec eingegeben werden. Die Amplitude ist frei wählbar

Step: Sprungfunktion. Erzeugt einen Signalsprung vom Wert ***Initial value*** zum Wert ***Final value*** zum Zeitpunkt ***Step time.***

Ramp: Erzeugt ein Signal mit einem rampenförmigen Anstieg. ***Start time*** ist i.a. 0, ***Slope*** gibt die Anstiegsgeschwindigkeit an. Bei einer Simulationszeit von z.B. 10 sek. läuft das Signal bei Slope=1 auf einen Endwert von 10, bei Slope=2.5 auf 25, usw.

Sine Wave: Sinusgenerator, die Frequenz (***Frequency***) muß in rad/sec und nicht in Hz eingegeben werden:

$$\omega\left[\frac{\text{rad}}{\text{sec}}\right] = 2\pi f.\left[\text{f in Hz}\right].$$

.

Die ***Amplitude*** ist frei einstellbar.

Chirp Signal: Erzeugt ein sinusförmiges Signal, das bei einer vorzugebenden unteren Frequenz (***Initial frquency***) beginnt und sich über eine vorzugebende Zeit (***Target time***) bis zu einer oberen vorzugebenden Frequenz (***Frequency at target time***) verändert. Im deutschsprachigen Raum wird ein solches System auch "Wobbelgenerator" genannt.

From Workspace: Mit Hilfe dieses Blockes lassen sich beliebige Signalform-Folgen "u" aus Matlab als Eingangssignal in Simulink-Strukturen einspeisen. In Matlab müssen ein Spalten-Zeitvektor "t" und ein Spalten-Signalvektor "u" generiert und anschließend zu einer Matrix mit zwei Spalten a=[t,u] zusammengefaßt werden. (Die Signalwerte "u" jeder Zeile der Matrix [t,u] müssen mit dem entsprechenden Zeitwerten "t" korrespondieren.) Trägt man den Matrixnamen a in die Rubrik ***Data:*** des ***From Workspace***-Blockes ein, steht diese Signalfolge dann in Simulink zur Verfügung. Falls die Simulations-schrittweite feiner gewählt wird als das Zeitraster in "t" wird zwischen den u-Werten automatisch linear interpoliert

Random Number: Erzeugt ein normalverteiltes Rauschsignal.

- **Sinks – Library**

Die Block-Library Sinks stellt Datensenken zur Verfügung mit denen Ausgangssignale aus Simulationsstrukturen grafisch darstellt und im Hauptspeicher oder auf einem externen Speicher abgelegt werden können.

Scope: Grafisches Oszilloskop (wurde bereits in einem vorangehnden Abschnitt ausführlich besprochen).

XY-Graph:Grafisches XY-Oszilloskop. Nähere Einzelheiten: ***Help***-Button

Display: Numerisches Display für Daten. Nähere Einzelheiten: ***Help***-Button

To Workspace: Kann alternativ zum Oszilloskop als Datenausgabe benutzt werden. Die Meßdaten werden im Matlab-Hauptspeicher unter dem Namen, der bei ***Variable name*** gewählt werden kann, abgelegt. Auf diese Daten kann mit Matlab-Befehlen in der Matlab Command Ebene zugegriffen werden.

- **Continuous – Library**

Die Continuous-Library stellt lineare, kontinuierliche Simulationselemente zur Verfügung. Darunter befinden sich sowohl Zeitbereichs-Modelle (Zustandsmodell) als auch s-Bereichs-Modelle (Integrator, verschiedene Übertragungsfunktions-formen). Die Parametrierung erfolgt mit der gleichen Syntax wie unter Matlab.

State-Space: Zustandsmodell. Eingabe der Parametermatrizen ***A,b,c,d*** und der Anfangswerte der Zustandsgrößen (***Initial conditions***) als Spaltenvektor mit n Elementen; n: Systemordnung.

Trasfer Fcn: Übertragungsfunktion in Polynomform. Eingabe des Zähler- (***Numerator***) und des Nennerpolynoms (***Denominator***).

Integrator: Bis auf den Anfangswert der Zustandsgröße (***Initial condition***) alle Defaulteinträge belassen.

Transport Delay: Totzeitglied mit der Totzeit "***Time delay***". Alle anderen Default-Einstellungen belassen.

- **Discrete – Library**

Die Discrete Library stellt zeitdiskrete Übertragungssystem-Elemente aus dem diskreten Zeitbereich (Zustandsmodell) und dem z-Bereich (Verzögerung um eine Abtastzeit, z-Übertragungsfunktion in Polynomform, usw.) zur Verfügung. Eine rekursive Differenzengleichung ist nicht implementiert, an ihrer Stelle wird eine z-Übertragungsfunktion oder ein zeitdiskretes Zustandsmodell genutzt

Die Konstruktion von Systemmodellen mit Mischungen aus Zeitbereichs- und auch z-Bereichs-Beschreibungen ist erlaubt. Auchh die Konstruktion von Systemmodellen

mit Mischungen aus Zeitbereichs-, Bildbereichs- und kontinuierlichen und zeitdiskreten Blöcken ist möglich.
Die Parametrierung erfolgt mit der gleichen Syntax wie unter Matlab.

<u>Achtung:</u>
Bei der Verknüpfung kontinuierlicher und zeitdiskreter Blöcke ist das Einfügen von Abtastgliedern zwischen kontinuierlichen und darauffolgenden zeitdiskreten Blöcken bzw. das Einfügen von Haltgliedern zwischen zeitdiskreten und darauffolgenden kontinuierlichen Blöcken ***nicht*** *notwendig.*
Simulink hat diese Funktionen direkt in die zeitdiskreten Simulationsblöcke implementiert.

Die in den Blöcken dieser Library abgefragte ***Sample time*** (die in allen Blöcken gleich groß eingetragen werden sollte) ist die Abtaszeit T des zeitdiskreten Systems.

Discrete Trasfer Fcn: z-Übertragungsfunktion in Polynomform. Eingabe des Zähler- (***Numerator***) und des Nennerpolynoms (***Denominator***) nach fallenden <u>positiven</u> Potenzen von z.

Unit Delay: Einheitsverzögerung um ein Abtastintervall T. Dient i.a. zur Simulation zeitdiskreter Systeme, die aus Verstärkern/ Abschwächern, Summierern/Subtrahierern und Unit Delays aufgebaut sind. Die ***Initial conditions*** sind die Anfangswerte der entsprechenden Zustandsgrößen. Sie werden i.a. Null gesetzt, wenn die Simulationsstruktur aus einer Übertragungsfunktion gewonnen wurde.

Discrete State-Space: Zeitdiskretes Zustandsmodell. Wurde in den theoretischen Lehrveranstaltungen nicht eingeführt und wird deshalb hier nicht genutzt.

- **Math-Library**

In der Math-Library befinden sich Blöcke für grundlegende mathematische Operationen zur Signalverarbeitung:

Sum: Summierer / Subtrahierer. Die Anzahl der mit + oder - zu verknüpfenden Signale wird in der ***List of signs*** angegeben. Es wird empfohlen, bei ***Icon shape*** mit rechteckigen Formen (***rectangular***) zu arbeiten

Gain: Verstärker / Abschwächer. Der gewünschte Verstärkungsfaktor wird bei ***Gain:*** eingetragen.

Math Function: frei wählbare mathematische Operationen (***exp, log, mod***, *usw.*).

Trigonometric Functions: Trigonometrische Funktionen (***sin, cos, usw***.)

Abs: Absolutwert-Bildner.

Sign: Vorzeichenerkenner.

- **Functions & Tables-Library**

Mit den Look-Up-Table-Funktionen können beliebig geformte statische Kennlinien und Kennflächen, die die Abhängigkeit einer Ausgangsgröße von zwei Eingangsgrößen beschreiben, erzeugt werden.

Look-Up Table: Bildet die Kennlinie des ***Vector of output values*** über dem ***Vector of input values***. Die Anzahl der Werte im Input- und im Outputvektor müssen gleich sein.

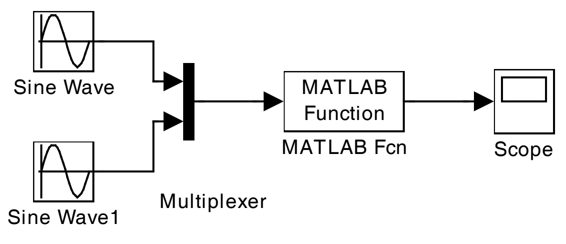
Look-Up Table (2D): Bildet die Kennfläche ***Table*** über den zwei Koordinaten ***Row*** und ***Column.*** Die Anzahl der Row- und Column-Werte soeir der Table-Werte müssen gleich groß sein.

Mit den Functions können Matlab-Operationen in Simulink genutzt werden:
Mit dem Block ***Fcn*** können alle mathematischen Operationen , wenn sie eine skalare Eingangs- und Ausgangs-Größe (zu einem Zeitpunkt) haben, durchgeführt werden. Die Eingangsgröße in diesen Block hat den festen Namen "u", die linke Seite der Zuweisungs-Anweisung (die Ausgangsgröße des Fcn-Blockes) wird weggelassen. So berechnet z.B. der Ausdruck "exp(u)+3" die Exponentialfunktion des Eingangssignals mit einem ständigen Offset von +3.

Der Block ***Matlab Fcn*** hat nicht die Einschränkung auf skalare Ein- und Ausgangs-größen. So berechnet z.B. der Ausdruck

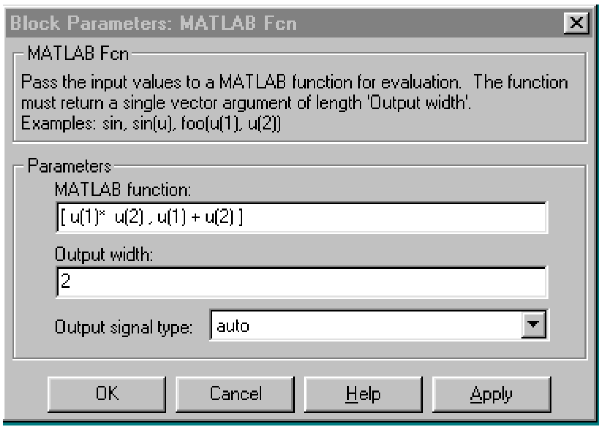
[u(1) * u(2) **,** u(1) + u(2)]

aus den zwei Eingangsgrößen u(1) und u(2) in der folgenden Simulationsstruktur

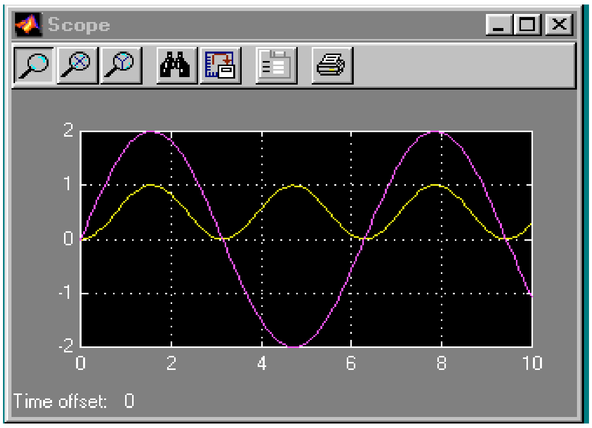


mit den folgenden (gleichen) Parametrierungen der ***Sine Wave Generatoren*** :

Amplitude: 1; ***Frequency***: 1; ***Phase***: 0; ***Sample time***: 0; und der Matlab Function



zeitlich parallel das Produkt und die Summe der beiden Eingangssignale (der Block ***Multiplexer*** wird etwas weiter hinten beschrieben):



Auch nicht in Matlab implementierte, sondern selbst geschriebene Matlab-Functions lassen sich mit Block ***Matlab Fcn*** in Simulink nutzen. Wir erläutern dies mit folgendem Beispiel:

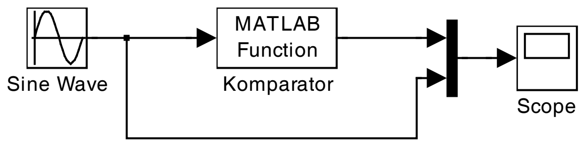
Es soll mit Hilfe des Matlab Fcn-Blocks ein Komparator entwickelt werden, der als Ausgangsgröße v = 0 ausgibt, wenn das Eingangssignal u >= 1 ist. Bei u < 1 soll v = 1 sein. Dazu schreiben wir (im Matlab-Editor) folgendes Function-File:

```
function [v] = komp (u)
if  u> = 1
     v = 0;
else v = 1;
end
```

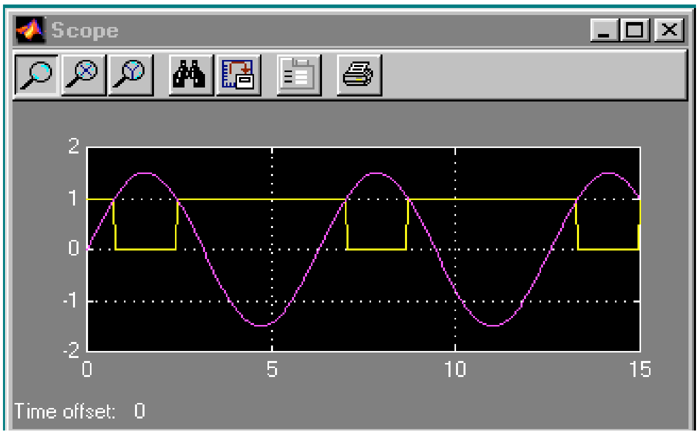
und speichern sie in unserem "work"-Verzeichnis unter dem Namen "komp.m".

Nach Rückkehr in die grafische Arbeitsoberfläche von Simulink ziehen wir uns den Matlab Function-Block in diese Ebene und öffnen ihn mit einem Doppelklick. In die

Zeile ***Matlab function*** tragen wir den Namen unserer erstellten Function "komp" ein und schließen den Parametrierungsblock. Bei der Verarbeitung von Skalaren geben wir bei "Output Width" eine **1** ein. Es muß nur darauf geachtet werden, daß das Eingangssignal in den Block wieder den festen Namen "u" hat.
Anschließend ändern wir die Blockunterschrift von "Matlab Fcn" in "Komparator". Die folgende Simulationsstruktur zeigt, daß der Komparator erwartungsgemäß arbeitet:



Die Simulationsstruktur "komp_test.mdl"



Dabei wurden folgende Simulationparameter angesetzt:

Simulations-Parameter:	***Start time***: 0; ***Stop time***: 15; ***Type:*** Fixed step; ***ode4 (Runge Kutta);*** ***Fixed step Size***: 0.05;
Sine Wave Generator:	***Amplitude***: 1.5; ***Frequency***: 1; ***Phase***: 0; ***Sample time***: 0
Scope Parameter :	***Time range***: auto; sonst Default-Einstellungen.

- **Nonlinear – Library**

Die Nonlinear Library stellt die wichtigsten nichtlinearen Simulationselemente zur Verfügung, zum Beispiel. Da wir uns primär mit linearen Systemen auseinander-setzen, sind die folgenden Simulationselemente nur kurz erläutert. Nähere Einzelheiten müssen in der Hilfe nachgelesen werden

Backslash: Bildet eine tote Zone mit Hysterese. Nähere physikalische Eigenschaften: Help-Button.

Dead Zone: Tote Zone ohne Hysterese. Tritt auf z.B. bei Getriebe-

Drehrichtungsumsteuerung.

Rate Limiter: Anstiegs- und Abfallzeitbegrenzer. Nähere Einzelheiten: Help-Button.

Relay: Zweipunktregler mit Hysterese.

Saturation: Signalbegrenzer, simuliert Geräteübersteuerungen.

Switch: Ereignisschalter, z.B. als Komparator einsetzbar.

Manual Switch: Mit der Maus umschaltbarer Schalter.

Coulomb & Viscose Friction: Simulation von Gleit- und Haft-Reibung.

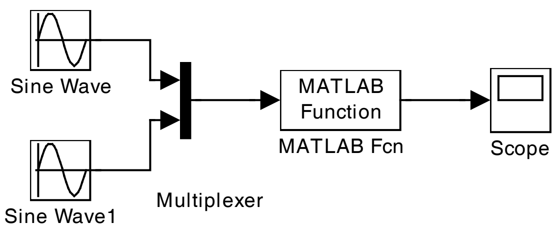
- **Signals & Systems-Library**

Aus dieser Library interessieren uns nur wenige Blöcke:

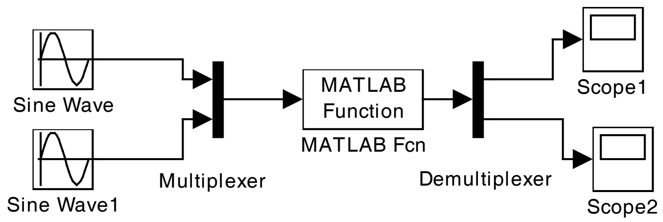
In1: Inport, dient z.B. zur Kennzeichnung (und damit zur Unterscheidung) von Signaleingangängen in ein Simulationsmodell, das mit ***Create Subsystem*** zu einem Symbol zusammengefaßt werden soll.

Out1: Outport: dient z.B. zur Kennzeichnung (und damit zur Unterscheidung) von Signalausgängen aus einem Simulationsmodell, das mit ***Create Subsystem*** zu einem Symbol zusammengefaßt werden soll.

Mux: Multiplexer: Dient zur grafischen "Busbildung" von Signalleitungen. Wird häufig benutzt, wenn z.B. mehrere Signale auf einem Scope angezeigt werden sollen oder über einen Blocks, der nur einen Eingang besitzt, zwei parallele Signale eingespeist werden müssen. Dies war z.B. der Fall bei einem vorangehenden Beispiel mit der Matlab Fcn:



Demux: Demultiplexer: Spaltet eine Signal-Leitung, die mehrere parallele Signale enthält in die einzelnen Signale auf. So können z.B. die beiden Signale der vorigen Abbildung mit einem Demultiplexer auf zwei Scopes abgebildet werden:



- **Blocksets & Toolboxes-Library**

Diese Library enthält die schon weiter vorn erwähneten Erweiterungs-Blocksets für Simulink, sie soll im Rahmen dieses Kurses nicht besprochen werden.

4.3 Schnittstellen zwischen Matlab und Simulink

Zwischen Matlab und Simulink bestehen verschiedene Schnittstellen, die es einerseits erlauben, Simulink-Modellstrukturen unter Matlab weitergehend zu analysieren (z.B. Übertragungsfunktionen oder Bodediagramme zu berechnen). Andererseits kann man durch andere Schnittstellenfunktionen mit Hilfe des Matlab-Befehlsatzes die Block-Library von Simulink erweitern oder Simulink-Strukturen von Matlab aus parametrieren.

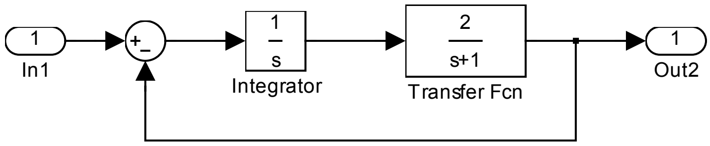
4.3.1 Der Übergang von Simulink nach Matlab

Mit der Einführung des Simulationsblockes ***To Workspace*** aus der ***Sinks Library*** haben wir schon einen wichtigen Vertreter für den Übergang aus der grafischen Arbeitsoberfläche in das Matlab Command Window kennengelernt: Unter Simulink erzeugte Datensätze von Signalverläufen konnten unter Matlab grafisch analysiert, gespeichert und als Eingangssignale in Matlab-Strukturen weiterverarbeitet werden. Mit ***To Workspace*** steht allerdings immer nur ein nichtparametrisches Modell im Zeitbereich unter Matlab zur Verfügung, ein Bodediagramm kann z.B. nicht aus diesen Daten berechnet werden.
Jedoch lassen sich auch parametrische Modelle in Form eines Zustandsmodells mit Hilfe des sehr mächtigen Befehls "Bilde ein lineares Modell":

[a, b, c, d] = **linmod** ('Simulinkfilename')

aus beliebig grafisch verknüpften Simulink-Modellen generieren. Das erzeugte Zustandsmodell kann dann unter Matlab beliebig weiterverarbeitet werden. Wir machen uns die Zusammenhänge an einem Beispiel klar:

Das folgende Simulink-Modell mit einem Inport als Eingang und einem Outport als Ausgang



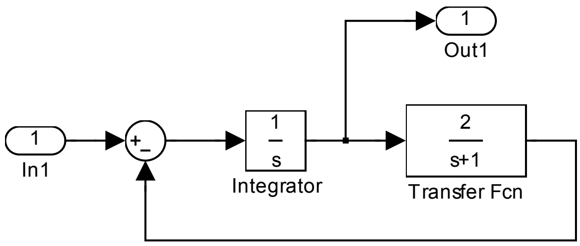
trägt den Filenamen "pt2.mdl". Der Befehlsaufruf **[** a, b, c, d **]** = **linmod (** '**pt2')** im Matlab Command Window erzeugt dann dort ein Zustandsmodell der Form

$$\underline{\dot{x}}(t) = \begin{bmatrix} -1 & 1 \\ -2 & 0 \end{bmatrix} \cdot \underline{x}(t) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \cdot u(t)$$

$$y(t) = \begin{bmatrix} 2 & 0 \end{bmatrix} \cdot \underline{x}(t)$$

das z.B. durch Aufruf der Befehlsfolge sys = **ss (**a,b,c,d**)**; **bode(**sys**)**; zum Zeichnen eines Bodediagramms benutzt werden kann.

Durch Verlagerung der In- und Outports, z.B. wie in dem folgenden Bild dargestellt, können weitere mathematische Zustandsmodelle gebildet werden, die das Übertragungsverhalten des Simulink-Modells zwischen den entsprechenden Ein- und Ausgängen modellieren.



Bekanntlich lassen sich Übertragungsfunktionen (und damit auch Frequenzgänge und Bodediagramme) nur von linearen Systemen angeben. Daher existiert auch kein spezieller Befehl, der eine nichtlineare Simulink-Struktur direkt in den Matlab-Bereich transformiert. <u>Der linmod-Befehl kann aber auch dazu benutzt werden, eine nichtlineare Simulink-Struktur zu linearisieren und dieses linearisierte Modell in der Matlab Command Ebene zur Verfügung zu stellen.</u>
Der linmod-Befehl muß dann um die Argumente "x" und "u" erweitert werden

[a, b, c, d] = **linmod** ('simulinkfilename', x, u)

Dabei sind x=[x_{1A}; x_{2A}; ... ; x_{nA}] der Zustandsvektor im Arbeitspunkt und u=u_A der Wert der Eingangsgröße im Arbeitspunkt. (Siehe /1/ "Linearisierung eines nichtlinearen Systems").

Auch diese Arbeitspunktwerte können mit Simulink leicht bestimmt werden. Am anschaulichsten geschieht dies durch Simulation des zu linearisierenden, nichtlinearen Systems mit $u(t) = \Delta u \cdot \sigma(t)$. Wobei Δu der Arbeitspunktwert u_A des Eingangssignals ist. Der Simulationszeitraum muß so gewählt werden, daß die Zustandsgrößen des Systems einen stationären Zustand erreichen. Diese stationären Endwerte sind dann die Arbeitspunktwerte $x = [x_{1A}; x_{2A}; ...; x_{nA}]$.

Matlab stellt auch den Befehl "trim" zur direkten Bestimmung der Arbeitspunktwerte zur Verfügung. Wegen der relativen Komplexität diese Befehls muß auf das Simulink-Handbuch verwiesen werden.

Für die Eingabe der Arbeitspunktwerte des Zustandsvektors x und eine Weiterarbeit mit dem erzeugten Zustandsmodell unter Matlab ist die Kenntnis der Reihenfolge der Zustände, wie sie Matlab aus dem Simulinkmodell entnimmt, wichtig. Sie kann nach Aufruf des linmod-Befehls mit folgender Kodefolge ermittelt werden:

```
[not_used,not_used,z]=simulinkfilename;
Zustandsreihenfolge=z
```

Ausgegeben wird der Spaltenvektor der Zustände, wobei immer der "simulinkfilename" und der Name der Zustandsgröße ausgegeben wird. Als Kennzeichnung der Zustandsgrößen wählt Matlab die Namen die unter den Dynamikblöcken stehen. In unserem Falle würde folgende Zustandsreihenfolge ausgegeben:

```
Zustandsreihenfolge =
     'pt2/Transfer Fcn'
     'pt2/Integrator'
```

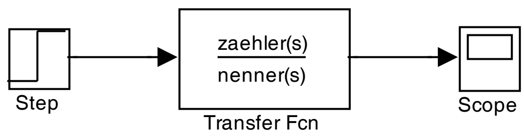
Das heißt, der erste Zustand des Zustandsmodell ist die Ausgangsgröße des in der Transfer Fcn befindlichen Integrators, der zweite die Ausgangsgröße des sichtbaren Integrators.
Enthält das Simulink-Modell Übertragungsfunktions-Blöcke höherer Ordnung (>2) ist die Zuordnung der Zustände (zumindest für diese Blöcke) nicht mehr eindeutig.

4.3.2 Der Übergang von Matlab nach Simulink

Mit Hilfe des ***From Workspace***-Blocks aus der ***Sources-Library*** und der ***Matlab Function***- und ***Fcn*** - Blöcke aus der ***Function & Tables*** - Library haben wir schon einige wichtige Übergänge von Matlab nach Simulink kennegelernt.

Wir wollen an dieser Stelle noch zeigen, daß sich Simulink-Simulationsstrukturen auch von Matlab her parametrieren lassen.

Trägt man in die Parametrierungsfenster der Simulink-Blöcke keine Zahlenwerte ein, sondern Variablennamen, kann man diesen mit einem Matlab-Programm Werte zuweisen. Öffnet man nun die Simulink-Simulationsstruktur und läßt dann das Matlab-Programm laufen, übernimmt die Simulink-Struktur diese Zuweisungen und kann mit ihnen gestartet werden.
Wir wollen uns die Zusammenhänge an einem Beispiel klar machen. In der folgenden Simulationsstruktur "test.mdl" sollen die Sprungantworten verschiedener Systemmodelle, die in Form ihrer Übertragungsfunktion vorliegen, mit verschiedenen Sprungamplituden berechnet werden. Die entsprechenden Änderungen sollen von einem Matlab-Programm vorgenommen werden.
Die Simulink-Simulationsstruktur habe folgende Form:



Bidl 4.2.10: Das Simulink-Modell "test.mdl"

Das Modell ist wie folgt parametriert:

Simulations Parameter:	***Start Time:*** 0; ***Stop Time***: 15; ***Type:*** Fixed-step;
	Fixed step size: 0.01***; ode4(Runge-Kutta).***
Step-Generator:	***Step time:*** 0; ***Initial value:*** 0; ***Final Value:*** amplitude;
Transfer Fcn:	***Numerator:*** zaehler; ***Denominator:*** nenner
Scope:	***Time Range***: auto; sonst Default-Einstellungen.

Wie man sieht, wurden beim Step-Generator an Stelle eines numerischen Wertes bei ***Final Value*** der Variablenname *amplitude* und bei ***Transfer Fcn*** an Stelle von Zähler- und Nennerpolynomvektoren bei ***Numerator*** der Variablenname *zaehler* und bei ***Denominator*** *nenner* eingetragen.

Diese Variablen können nun von Matlab her parametriert werden:

```
% Matlab-Programm zur Parametrierung des
% Simulink-Modells "test.mdl"
%
test                % öffnet das Simulinkmodell
amplitude = 2.5;
zaehler = 1.5;
nenner = [1, 0.2, 1];
```

Läßt man nun dieses Matlab-Programm laufen, öffnet der erste Beehl die zu parametrierende Simulink-Struktur "test.mdl", falls sie nicht schon offen ist. Die restlichen Befehle nehmen die oben gennaten Parametrierungen der Simulink-Blöcke vor. Diese Prametrierungsmöglichkeit wird häufig dann genutzt, wenn mit einem umfangreichen Matlabprogramm z.B. die Koeffizienten der Übertragungsfunktion eines digitalen Filters oder Reglers in Matlab berechnet und die Funktion des Filters oder Reglers in Simulink getestet werden soll.

Ähnliche Parametrierungen lassen sich mit dem Befehl

```
set_param ( 'Filename/Blockname','Parametername' , ...
          'Parameter','Parametername','Parameter', ... )
```

durchführen. Mit ihm ist es sogar möglich Parameter zu setzen, die in der Simulink-Struktur nur aus einer Liste gewählt werden können. Wegen der relativen Kompliziertheit diese Befehls, wollen wir hier nicht weiter darauf eingehen. Nähere Erläuterungen finden sich in der Matlab-Hilfe

Zwei weitere interessante Befehle zur Steuerung der Simulations-Randbedingungen sind die Befehle ***sim*** und ***simset***. Auch auf sie soll aus Zeitgründen nicht eingegangen werden.

Literatur

/1/	M. Ottens	"Grundlagen der Systemtheorie" TFH-Berlin, Fachbereich VI (Informatik+Medien)
/2/	M. Ottens	"Einführung in die Regelungstechnik" TFH-Berlin, Fachbereich VI (Informatik+Medien)
/3/	Ohne Autorenangabe	"Matlab" Users Guide, "Matlab" Reference Guide Matlab Version 5.3 (R11) The MathWorks, Inc., Natick, Mass., USA, 1999
/4/	Ohne Autorenangabe	"Simulink" Users Guide Simulink Version 3 (R11) The MathWorks, Inc., Natick, Mass., USA, 1999
/5/	www.mathworks.com www.mathworks.de	
/6/	M. Ottens	"Einführung in die Nutzung des Realtime- Workshops des CAE-Programms Matlab" TFH-Berlin, Fachbereich VI (Informatik+Medien)
/7/	A.Angermann, u.a.	"Matlab-Simulink-Stateflow" Grundlagen, Toolboxen, Beispiele Oldenbourg Verlag, München, Wien, 2003
/8/	Ohne Autorenangabe	"Buildung GUIs with Matlab" Version 5, The MathWorks, Inc., Natik, Mass., USA, 1997