TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik, Institut für Angewandte Informatik, Professur Technische Informationssysteme

EINFÜHRUNG IN MATLAB / SIMULINK

Erstellt von: Dipl.-Inf. Holger Preiß

Überarbeitet von: Dr.-Ing. Alexander Dementjev

ALLGEMEINE BEMERKUNGEN

Sollten Sie Verbesserungsvorschläge, Fragen oder Probleme haben, so wenden Sie sich am besten persönlich an Ihren Betreuer oder schreiben Ihm eine E-Mail (vorname.nachname@inf.tu-dresden.de).

EINFÜHRUNG IN MATLAB

EINLEITUNG

Zur Verbesserung der Lesbarkeit werden in dieser Anleitung folgende Konventionen verwendet:

Hervorhebung	-	wichtige Informationen werden durch normalen **Fettdruck** hervorgehoben
„Workspace“	-	Englische Bezeichnungen werden in Anführungszeichen eingebettet
`plot(n,x);`	-	Kommandos werden in `Courier New 11 pt` dargestellt
„File/New“	-	auf Menüs, Menübefehle und auf Namen von Dialog-Boxen wird in *kursiver Schrift* verwiesen
signal.m	-	Verzeichnis- oder Dateinamen werden durch die Schriftart ***Arial fett kursiv 11 pt*** gekennzeichnet

Wenn eine Bezeichnung (nach Meinung des Autors) sowohl in deutscher als auch in englischer Sprache sinnvoll zu verwenden ist, wird hier die deutsche Version bevorzugt.

MATLAB ist eine technisch-wissenschaftliche Software für leistungsstarke numerische Berechnungen und professionelle Visualisierung von Daten und Ergebnissen. Auch im deutschen Sprachraum erfährt MATLAB an Hochschulen und in der Industrie eine zunehmende Verbreitung. Wichtige Charakteristika dieses Programms sind u. a. die unmittelbaren graphischen Darstellungsmöglichkeiten, eine große Vielfalt eingebauter Funktionen, die Möglichkeit des Hinzufügens eigener Funktionen und eine sehr einfache Programmierung.

Dabei ist MATLAB besonders für die Berechnung und Simulation von Systemen der digitalen Signalverarbeitung geeignet. Ziel dieser Einführung ist es, dem Leser eine benutzerorientierte Beschreibung des Programmpakets MATLAB zu geben.

Dabei sollen die Fragen

1 was ist MATLAB,
2 wie arbeitet man mit MATLAB und
3 wie wird MATLAB in der Signalverarbeitung eingesetzt?

beantwortet werden. Insbesondere soll, ohne die MATLAB-Handbücher ersetzen zu wollen, soviel spezifische Information zusammengestellt werden, dass der Leser in die Lage versetzt wird, selbständig einen Einstieg in MATLAB/Simulink zu finden.

STATISCHER AUFBAU

Das Blockdiagramm nach Bild 1 soll die Funktionseinheiten von MATLAB veranschaulichen, so wie sie sich dem Benutzer darstellen. Das Bild dient zur Beschreibung des statischen Aufbaus. Den Kern von MATLAB bilden das „MATLAB Command Window“ sowie zwei weitere Einheiten: der Interpreter, der sich mit dem Bereitschaftszeichen meldet, und der „Command Line Editor“, der es ermöglicht, über die Tastatur Eingaben zu machen. Seit MATLAB Release 12 wurde das „Command Window“ in eine integrierte Entwicklungsumgebung eingebettet, die, beliebig konfigurierbar, den Zugriff auf ein “Launch Pad” (bequemes Starten von Programmen), die Variablen des „Workspace“ (mit integriertem „Workspace Editor“), das aktuelle Arbeitsverzeichnis und auf eine „Command History” (bequemer Aufruf bereits gegebener Kommandos mittels Mausklick) bietet.

Zur Durchführung von Berechnungen bietet MATLAB als Interpretersprache zwei Betriebsweisen:

1 „**Command Driven Mode“:** Mit dem Command Line Editor werden im „MATLAB Command Window“ Anweisungen eingegeben. Das Abschließen der Anweisung mit der Return-Taste veranlasst dann die sofortige Ausführung der Anweisung.
2 „**File Driven Mode“:** Mit einem externen Texteditor wird ein MATLAB-Programm als Folge von MATLAB- Anweisungen geschrieben und unter einem Namen mit der Endung ***.m*** abgespeichert. Wird dann im „MATLAB Command Window“ dieser Name eingegeben und mit der Return-Taste abgeschlossen, so wird das Programm gestartet und damit die Anweisungsfolge des Programms ausgeführt.

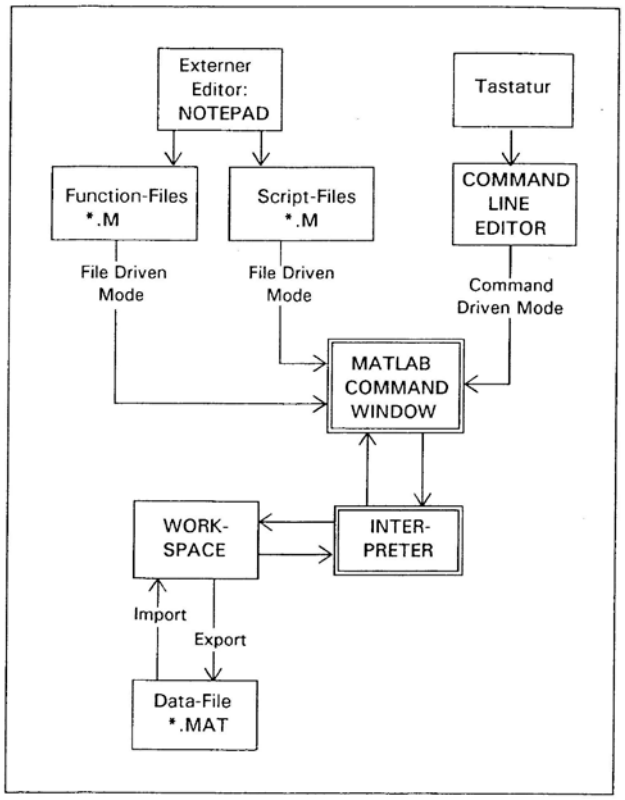

Bild 1 Statischer Aufbau von MATLAB

Der Interpreter arbeitet mit einem „Workspace“ genannten Speicherbereich zusammen, in dem alle Variablen gespeichert werden. Dabei besteht die Möglichkeit, den Inhalt des „Workspace“ in einer externen Datei mit der Endung ***.mat*** oder als Datei im ASCII-Format abzuspeichern und von dort auch wieder zu laden.

Programme

Mit dem in Bild 1 angeführten Texteditor lassen sich auf zwei verschiedene Weisen Programme schreiben: „**Script Files**“ und „**Function Files**“. Beide werden in Dateien mit der Endung ***.m*** abgespeichert. Damit auf diese Dateien bzw. Programme zurückgegriffen werden kann, muss MATLAB der Pfad, auf dem die Dateien zu erreichen sind, mitgeteilt werden.

Dies kann auf drei Weisen geschehen:

1. durch direktes Eintragen des Pfades in die Datei ***$(MATLAB)\toolbox\local\pathdef.m*** mit Hilfe eines Texteditors
2. mit Hilfe des **Path-Befehls** vom „MATLAB Command Window“ aus
3. über das Menü „*File/Set Path...*“ der integrierten Entwicklungsumgebung

Script-Files

Diese Programme werden mit ihrem Namen vom „MATLAB Command Window“ aufgerufen. Dabei handelt es sich um Folgen von MATLAB-Anweisungen, die beim Aufruf nacheinander ausgeführt werden. Betrachtet werde z. B. die Script-Datei ***signal.m*** :

```
% Script File signal.m

n = 1:100;                  % Erzeugung eines Zeilenvektors
x = sin(2*pi*0.01*n);       % Erzeugung einer Sinusfolge
plot(n,x);                  % graphische Darstellung der Sinusfolge
```

Bei der Eingabe von `signal` im „Command Window“ werden die 3 Befehle ausgeführt. Die Variablen `n` und `x` bleiben im „Workspace“ gespeichert. In MATLAB modellierte Systeme sind Script-Dateien.

Function-Files

MATLAB verfügt über einen Vorrat intern implementierter Funktionen. Dazu gehören die Kreisfunktionen, die Hyperbelfunktionen, die Logarithmusfunktion usw. Von entscheidender Bedeutung ist nun, dass mit Hilfe der „Function Files“ dem System benutzerdefinierte Funktionen hinzugefügt werden können. Dazu werde das folgende Beispiel zur Berechnung des Mittelwerts eines Zeilenvektors `x` betrachtet:

```
% Funktion zur Ermittlung des Mittelwerts eines Signals

function y = mittel(x);

  N=length(x);
  y=sum(x)/N;
```

Am Anfang der Datei kann ein Kommentar zur Verwendung der definierten Funktion geschrieben werden, den ein Anwender durch „`help Funktionsname`" aufrufen kann. Im Beispiel liefert also der Aufruf „`help mittel`" die Ausschrift „`Funktion zur Ermittlung des Mittelwerts eines Signals`".

Die zweite Zeile kennzeichnet die Datei durch das Schlüsselwort `function` als „Function File". Ferner enthält sie die Eingabe- und Ausgabeparameter sowie den Funktionsnamen. In der letzten Zeile wird der berechnete Wert dem Ausgabeparameter zugewiesen. Wichtig ist, dass die in einem „Function File" verwendeten Variablen nur lokal bekannt sind.

Abgespeichert wird der Programmtext in einer Datei mit der Endung ***.m***, wobei Dateiname und Funktionsname übereinstimmen. Befindet sich die Datei in einem MATLAB bekannten Verzeichnis, so ist neben der Funktion `sin(x)` fortan auch eine Funktion `mittel(x)` verfügbar. Auf diese Weise kann der Vorrat an benutzerdefinierten Funktionen beliebig erweitert werden. Für verschiedene Anwendungsbereiche wurden Sammlungen von „Function Files" in sogenannten „Tool Boxes" zusammengefasst, z. B. in der „Signal Processing Tool Box".

PROGRAMMIERSPRACHE

Die Bezeichnung MATLAB leitet sich von **Matrix Laboratorium** her; denn die Basisgröße in MATLAB ist die Matrix. MATLAB ist eine höhere Programmiersprache auf der Basis eines Interpreters. Der Operand für die Programmiersprache MATLAB ist die Matrix und ihre Sonderfälle: Skalar, Spaltenvektor und Zeilenvektor. Die Elemente der Matrizen sind reelle Zahlen, komplexe Zahlen oder Ausdrücke aus reellen oder komplexen Zahlen. Ausdrücke entstehen durch Verknüpfung von Matrizen mit Hilfe von Operatoren. Gespeichert werden die Matrizen in Variablen. Insgesamt ist MATLAB eine FORTRAN-ähnliche Programmiersprache. Im folgenden werden einige, besonders häufig vorkommende Details zusammengestellt.

Operatoren

Tabelle 1 MATLAB-Operatoren und deren Schreibweise

OPERATOREN							
Matrix		**Array**		**relational**		**logisch**	
+	Addition	+	Addition	**<**	kleiner	&	UND
-	Subtraktion	-	Subtraktion	**<=**	kleiner gleich	**|**	ODER
*****	Multiplikation	**.***	Multiplikation	**>**	größer	~	NICHT
/	rechtsseitige Division	**./**	rechtsseitige Division	**>=**	größer gleich		
****	linksseitige Division	**.**	linksseitige Division	**==**	gleich		
^	Potenz	**.^**	Potenz	**~=**	ungleich		
‘	transponiert						

Tabelle 1 gibt eine Übersicht über die in MATLAB verfügbaren Operatoren, mit denen mathematische Ausdrücke gebildet werden können. Es werden Matrix-, Array-, relationale und logische Operatoren unterschieden. So ist es z. B. bei der Multiplikation wichtig, den Matrizencharakter der Variablen zu beachten. Neben der Matrizenmultiplikation gibt es auch die Multiplikation als Array-Operation, bei

der die Elemente beider Matrizen mit gleicher Indizierung miteinander multipliziert werden.

Die folgenden beiden MATLAB-Zeilen zur Erzeugung einer amplitudenmodulierten Sinusfolge enthalten eine Multiplikation als Array-Operation.

```
n=1:100;
x=(1+0.5*sin(2*pi*0.05*n).*sin(2*pi*0.25*n));
```

Matrizen und Vektoren

Einige wichtige Grundregeln sind:

1 Jede Variable ist eine Matrix.
2 Es gibt keine Variablen Deklaration.
3 Es gibt keine Matrix-Dimensionierung; sie erfolgt automatisch durch Wertzuweisung.
4 Bei den Variablen-Namen werden große und kleine Buchstaben unterschieden.

Eine Matrix kann im „MATLAB Command Window“ manuell mit einer Wertzuweisung in eckigen Klammern eingegeben werden, z. B.:

```
A = [l 2 3 4 5; 6 7 8 9 10; 11 12 13 14 15];
```

Die Trennzeichen zwischen den Elementen einer Zeile sind Leerzeichen oder Kommata; als Trennzeichen zwischen den Zeilen dient das Semikolon oder ein CR (carriage return). Der Abschluss einer MATLAB-Anweisung mit einem Semikolon unterdrückt die sonst automatisch erfolgende Ausgabe auf dem Bildschirm. Nach der Eingabe der Matrix A erfolgt die Speicherung im „Workspace“. Die Matrix lässt sich durch die Eingabe des Variablen-Namens wieder ausgeben. Man erhält:

```
A=  1  2  3  4  5

    6  7  8  9  10

   11 12 13 14  15
```

Häufig wird die Erzeugung eines Zeilenvektors erforderlich. Drei Verfahren seien angeführt. Eine Eingabesyntax lautet:

```
Z=[<Anfangswert>: <Inkrement>: <Endwert>];
```

So ergibt die Eingabe von `Z=[2:3:11]` den Zeilenvektor `Z=[2 5 8 11]`.

Benötigt man Vektoren oder Matrizen mit Nullen oder Einsen, so können sie mit den Funktionen `zeros` oder `ones` erzeugt werden. Die Syntax ist:

```
B = zeros(<Zeilenanzahl>,<Spaltenanzahl>);

C = ones(<Zeilenanzahl>,<Spaltenanzahl>);
```

So liefert `zeros(1,6)` einen Zeilenvektor mit 6 Nullen.
Möglich ist die Indizierung eines Elements der Matrix oder einer Teilmatrix. So liefert die Eingabe

von `A(2,3)` aus dem obigen Beispiel den Wert `8`und von `A(2:3,3:4)`die Teilmatrix

```
 8  9
13 14
```

Die Werte in Vektoren und Matrizen können auch komplex sein. Ein Imaginärteil wird durch den nachgestellten Bezeichner i oder j gekennzeichnet. So wird durch die Anweisung `x=5i` der Variable `x` der imaginäre Wert `5` zugewiesen. Eine komplexe Wertzuweisung erfolgt durch eine Addition von Realteil und bezeichnetem Imaginärteil, z. B. durch `x=3+5i`.

Schleifen

Natürlich kennt MATLAB, wie die anderen Programmiersprachen auch, eine Laufanweisung für die wiederholte Ausführung einer Folge von Anweisungen. Sie hat die Form

```
for i=1:n, <Folge von Anweisungen>, end;
```

Die wiederholte Ausführung wird ebenfalls mit Hilfe einer while-Schleife erreicht, in der eine Folge von Anweisungen solange wiederholt wird, bis eine Bedingung erfüllt ist. Sie hat die Form

```
while <Bedingung>, <Folge von Anweisungen>, end;
```

Ein- und Ausgabe

Hierbei geht es um Eingaben von der Tastatur und Ausgaben auf dem Bildschirm. Hier seien zwei häufig verwendete Konstruktionen angeführt: `input` und `disp`.

Die Input-Anweisung ermöglicht es, während des Programmlaufs bestimmten Variablen Werte zuzuweisen. Dabei wird der Programmlauf unterbrochen, über die Tastatur eine Eingabe vorgenommen und diese einer Variablen zugewiesen. Die Anwendung zeigt das folgende Beispiel:

```
x=input('Der Parameter x=');
```

Die disp-Anweisung ermöglicht es, während des Programmlaufs Meldungen auf den Bildschirm zu bringen. Die Anwendung zeigt das folgende Beispiel:

```
disp('Dies ist eine Meldung für den Bildschirm');
```

FUNKTIONEN ZUR SIGNALANALYSE

Bei der Untersuchung von Systemen der Signalverarbeitung sind Funktionen zur Analyse von Signalen unerlässlich. Signalverarbeitung bedeutet, dass Signale verarbeitet, d. h. verändert werden. Daher sind die Signale am Eingang und am Ausgang eines Systems zu analysieren. Die wichtigsten Analysemethoden sind das Oszillogramm und das Spektrum. Daher sollen die beiden wichtigsten, dabei zur Anwendung kommenden MATLAB-Funktionen näher betrachtet werden.

Oszillogramm: PLOT

Besondere Stärken hat MATLAB bei der Visualisierung von Signalen, Spektren und Frequenzgängen. Die Werte dieser Größen können als Zeilen- oder als Spaltenvektor vorliegen.

Die MATLAB-Funktion `plot` ist dabei der zentrale Befehl. Seine Syntax kann wie folgt angegeben werden.

```
plot([xmin],[ymin],[xmax],[ymax],[x],y,['df']);
```

Darin sind:

`xmin,ymin`		Koordinaten des linken unteren Eckpunktes des Plotfensters
`xmax,ymax`		Koordinaten des rechten oberen Eckpunktes des Plotfensters
`x`		Zeilenvektor für die Abszisse
`y`		Zeilenvektor für die Ordinate, d.h., das Signal
`d`		Zeichen, die festlegen, ob die Signalwerte als Punkte dargestellt werden oder durch Linien miteinander verbunden werden
	Linien:	d ist `-/--/:/-.`
	Punkte:	d ist `./+/*/o/x`
`f`		Zeichen zur Festlegung der Farbe: `r/g/b/y`

Bis auf den Zeilenvektor des Signals sind alle Angaben optional.

Spektrum: FFT

Die Fast Fourier Transformation, abgekürzt FFT, ist ein effizientes Verfahren zur Berechnung der Diskreten Fourier Transformation, d. h. der DFT-Summe. Die DFT einer, i. a. komplexen Zahlenfolge $\underline{x}(n)$ mit N Folgenelementen

$$\underline{y}(m) = \frac{1}{N}\sum_{n=0}^{N-1}\underline{x}(n)\cdot e^{-j2\pi\frac{mn}{N}} \qquad (1)$$

wird durch die MATLAB-Funktion FFT mit zwei Abwandlungen folgendermaßen realisiert:

$$\underline{y}(m+1) = \sum_{n=0}^{N-1}\underline{x}(n+1)\cdot e^{-j2\pi\frac{mn}{N}} \qquad (2)$$

Der Faktor `1/N` fehlt in der MATLAB-FFT; er taucht bei der inversen DFT auf. Zusätzlich tritt ein Versatz bei der Indizierung auf; dies ist notwendig, weil bei der Indizierung von Matrizen die Null nicht verwendet wird. Damit ergibt sich die folgende Zuordnung zwischen dem Index des Frequenzvektors $\underline{y}(i)$ und der bezogenen Frequenz

$$f' = \frac{i-1}{N} \qquad (3)$$

Dies ist bei der Darstellung des Spektrums zu berücksichtigen. Die DFT-Formel nach Gl. (1) ergibt bei der Analyse einer Sinusfolge eine Spektrallinie mit der Höhe des halben Scheitelwerts. Somit sind bei der Programmierung der Faktor 1/N, der Faktor 2 und der Versatz bei der Indizierung zu berücksichtigen. Die Syntax der FFT-Funktion ergibt sich aus:

```
<Frequenzvektor>=fft(<Folgenvektor>,<Anzahl der Folgenelemente>);
```

SIGNALVERARBEITUNG

Die Anwendungsgebiete von MATLAB sind sehr weit gestreut; daher sollen hier einige spezifische Hinweise zum Einsatz von MATLAB in der digitalen Signalverarbeitung zusammengestellt werden.

Signale

Eindimensionale Signale der Nachrichtenübertragung sind in MATLAB Vektoren, d. h. Matrizen mit nur einer Zeile oder nur einer Spalte. Was bei dem **digitalen Signaltyp** offensichtlich ist, nämlich dass die Signale in MATLAB als Vektoren darzustellende Zahlenfolgen sind, bedarf bei dem **analogen Signaltyp** einiger Anmerkungen. Analoge Signale sind kontinuierliche Zeitfunktionen; die Verarbeitung dieser Signale in MATLAB erfordert natürlich ihre Darstellung ebenfalls durch Zahlenfolgen.

Systeme: Simulation

Ein lineares Übertragungssystem für Zahlenfolgen kann durch eine Übertragungsfunktion

$$H(z)=\frac{Y(z)}{X(z)}=\frac{A_0+A_1\cdot z^{-1}+A_2\cdot z^{-2}+...+A_M\cdot z^{-M}}{1+B_1\cdot z^{-1}+B_2\cdot z^{-2}+...+B_N\cdot z^{-N}}$$

dargestellt werden. Eine Implementierung dieses Systems kann in MATLAB mit Hilfe der Funktion `filter` erfolgen. Diese Funktion berechnet die Elemente der Ausgangsfolge `y` aus den Elementen der Eingangsfolge `x` und benötigt dafür die Koeffizienten der Übertragungsfunktion H(z). Eingangs- und Ausgangssignal sowie Zähler- und Nennerkoeffizienten sind Zeilenvektoren. Da bei Vektoren und Matrizen der Index Null nicht verwendet wird, ergibt sich eine abgewandelte Schreibweise:

$$H(z)=\frac{Y(z)}{X(z)}=\frac{A(1)+A(2)\cdot z^{-1}+A(3)\cdot z^{-2}+...+A(M)\cdot z^{-M}}{1+B(2)\cdot z^{-1}+B(3)\cdot z^{-2}+...+B(N)\cdot z^{-N}}$$

Das folgende MATLAB-Beispielprogramm berechnet eine Sinusfolge der Frequenz `f0=0.01` mit 50 Folgenelementen. Die Koeffizienten eines Butterworth-Tiefpasses mit der Grenzfrequenz `fg=0.05` werden ermittelt. Darauf errechnet die Funktion `filter` das Ausgangssignal `y`.

```
fo=0.01;
fg=0.05;
[A,B]=butter(4,2*fg);
n=1:50;
x=sin(2*pi*fo*n);
y=filter(A,B,x);
```

Systeme: Berechnung

Zur Berechnung von Systemen der digitalen Signalverarbeitung, die sich auf eine Übertragungsfunktion H(z) zurückführen lassen, stellt MATLAB eine Reihe von sehr leistungsfähigen „MATLAB Function Files“ zur Verfügung. Die folgende Tabelle zeigt eine Zusammenstellung der für Systemberechnungen wichtigsten „Function Files“.

Tabelle 2 „MATLAB Function Files“ für Systemberechnungen

DIGITALE SIGNALVERARBEITUNG		
Anwendungsbereich	**M-File**	**Beschreibung**
REKURSIVE FILTER Dimensionierung derÜbertragungsfunktion H(z)	bilinear	Ermittlung aus F(s)
	butter	Butterworth - Filter
	cheb1	Chebyshev - Tiefpass Typ 1
	cheby2	Chebyshev - Tiefpass Typ2
	Ellip	Cauer - Tiefpass
TRANSVERSALE FILTER Dimensionierung derÜbertragungsfunktion H(z)	fir1	Fenster-Methode
	fir2	Fenster-Methode
	remez	Parks Mc Clellan
BERECHNUNG aus Übertragungsfunktion H(z)	freqz	Amplitudengang, Phasengang
	conv	Faltung zweier Folgen
	grpdelay	Gruppenlaufzeitgang

Bei den Berechnungen können zwei Bereiche unterschieden werden:

1 Berechnung der Koeffizienten der Übertragungsfunktion aus gegebenen Anforderungen und
2 Berechnung von Amplitudengang, Phasengang und Gruppenlaufzeitgang aus den Koeffizienten der Übertragungsfunktion.

EINFÜHRUNG IN SIMULINK

EINLEITUNG

Simulink ist ein graphisch orientiertes Programm zur Simulation von allgemeinen, dynamischen Systemen. Zusammen mit MATLAB bietet Simulink eine komplette, integrierte Arbeitsumgebung für Analyse, Modellierung, Visualisierung und Simulation von dynamischen Systemen an. Es eignet sich sehr gut für die Funktionssimulation nachrichtentechnischer Systeme.

Folgende Systemtypen werden von Simulink unterstützt: lineare, nichtlineare, diskrete, kontinuierliche und hybride. Für die Modellbildung von Systemen stehen drei verschiedene Möglichkeiten zur Auswahl:

- Graphisch-blockorientierte Arbeitsweise,
- das Programmieren in MATLAB und
- die C- oder FORTRAN-Programmierung.

Ziel dieser Einführung ist es, dem Leser eine benutzerorientierte Beschreibung des Programmpakets Simulink zu geben. Dabei sollen die Fragen

- Was ist Simulink, wie arbeitet man mit Simulink? und
- Wie wird Simulink in der Signalverarbeitung eingesetzt?

beantwortet werden. Die Beschreibung soll nicht die Handbücher ersetzen, sondern hauptsächlich praktische Hinweise geben. Eine vollständige Beschreibung dieses, sehr mächtigen Werkzeuges ist also in keiner Weise beabsichtigt.

STATISCHER AUFBAU

Zunächst soll der statische Aufbau von Simulink erläutert werden. Als grafische Oberfläche spielen dabei verschiedene Fenster eine wichtige Rolle, deren Bedeutung erklärt werden soll.

Übersicht

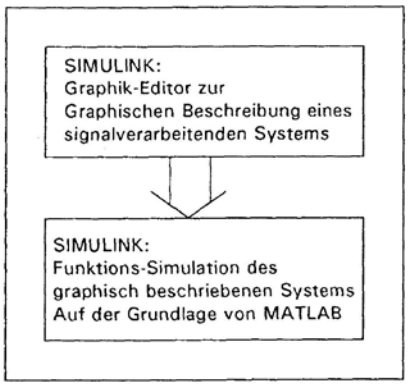

Bild 1 Arbeitsschritte und Werkzeuge in Simulink

Der Graphik-Editor ermöglicht die graphische Beschreibung von Systemen der digitalen und analogen Signalverarbeitung. Durch die Verbindung mit MATLAB ist die Simulation solcher Systeme möglich. Bild 1 zeigt die grundsätzlichen zwei Arbeitsschritte im Umgang mit Simulink.

Das mit Hilfe des Graphik-Editors beschriebene System ermöglicht eine Simulation durch die Inanspruchnahme von MATLAB. Das Zusammenwirken der einzelnen Blöcke und Fenster beim Arbeiten mit Simulink zeigt Bild 2 und Bild 3 zeigt das Zusammenspiel der einzelnen Fenster.

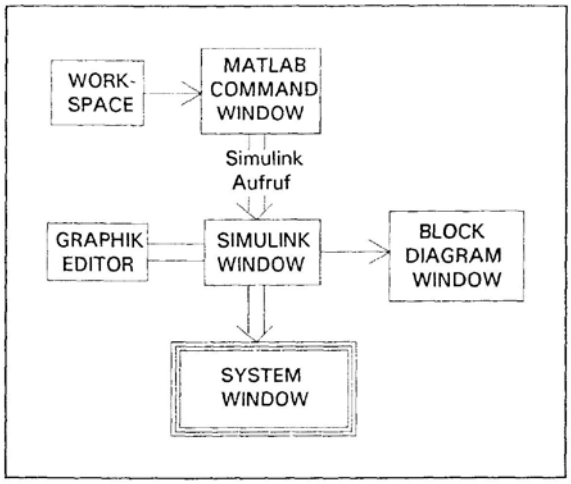

Bild 2 Statischer Aufbau von Simulink

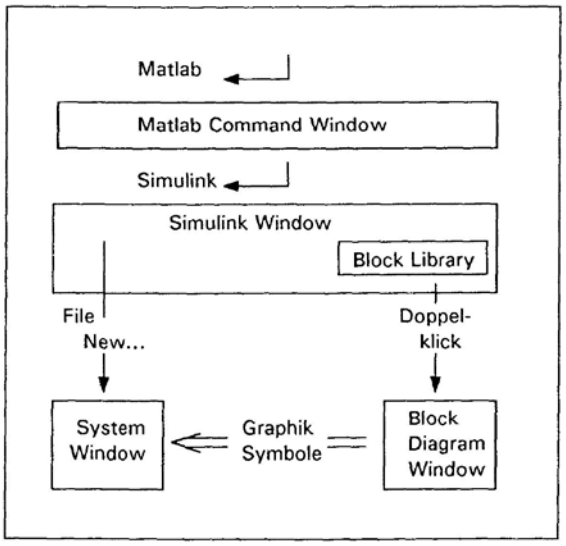

Bild 3 Zusammenhang der Simulink Fenster

Vom „MATLAB Command Window“ aus wird das Programm **Simulink** mit dem Befehl `simulink` (in Release 12 mit `simulink3`) gestartet. Daraufhin erscheint das Simulink-Bibliotheksfenster, das die „Simulink Block Library“ enthält.

Vom Simulink-Fenster aus führt der Weg zu zwei weiteren Fenstern. Über den Menü-Punkt „*File/New/Model*“ wird das Modellfenster geöffnet. In diesem Fenster findet die Modellierung von Systemen der digitalen und analogen Signalverarbeitung statt.
Durch einen Maus-Doppelklick auf einen Block der „Simulink Block Library“ öffnet sich ein weiteres Bibliotheksfenster mit einem Vorrat an Graphik-Symbolen. Das Editieren eines Systems erfolgt dann prinzipiell so, dass Graphik-Symbole durch Ziehen mit der Maus in das Modellfenster übernommen werden und dort zu Systemen zusammengefügt werden.

Simulink-Fenster

Das Simulink-Bibliotheksfenster enthält wie auch alle von hier geöffneten Fenster eine Menüleiste mit den Punkten *File*, *Edit*, *View*, *Format* und *Help*.

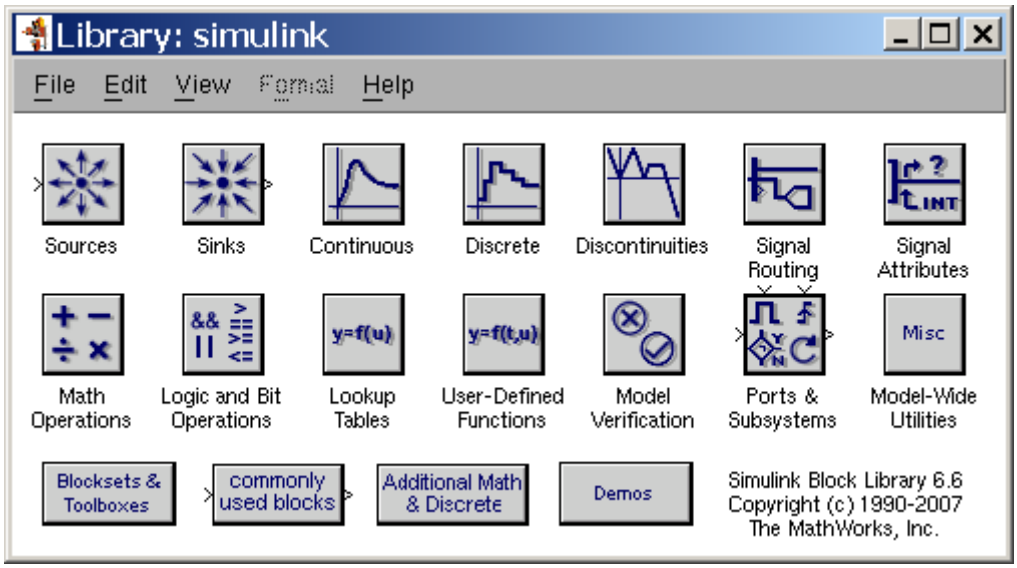

Bild 4 Simulink Bibliotheksfenster

Im folgenden wird nur der Menü-Punkt *File* angesprochen. Er ist eine Zusammenfassung der dateibezogenen Operationen. Zunächst werden die beiden Punkte *New* und *Open* betrachtet. Über *New/Model* wird ein Modellfenster geöffnet, in dem ein neues System modelliert wird. Nach der Modellierung kann es über *Save* als Datei unter einem Namen mit der Endung ****.mdl*** abgespeichert werden. Der Punkt *Open* ermöglicht das erneute Laden von Modellen mit bereits erstellten Systemen.

Bibliotheksfenster

Durch einen Doppelklick auf ein Symbol der „Simulink Block Library“ öffnet sich ein weiteres Bibliotheksfenster, wie es z. B. Bild 5 für die Gruppe „*Discrete*“ zeigt. Dieses Fenster gestattet jetzt den Zugriff auf einzelne Blöcke dieser Bibliothek und/ oder ermöglicht den Zugang zu weiteren Unterbibliotheken. Es sind somit beliebige Hierarchien möglich, die auch nachträglich durch jeden Anwender erstellt werden können.

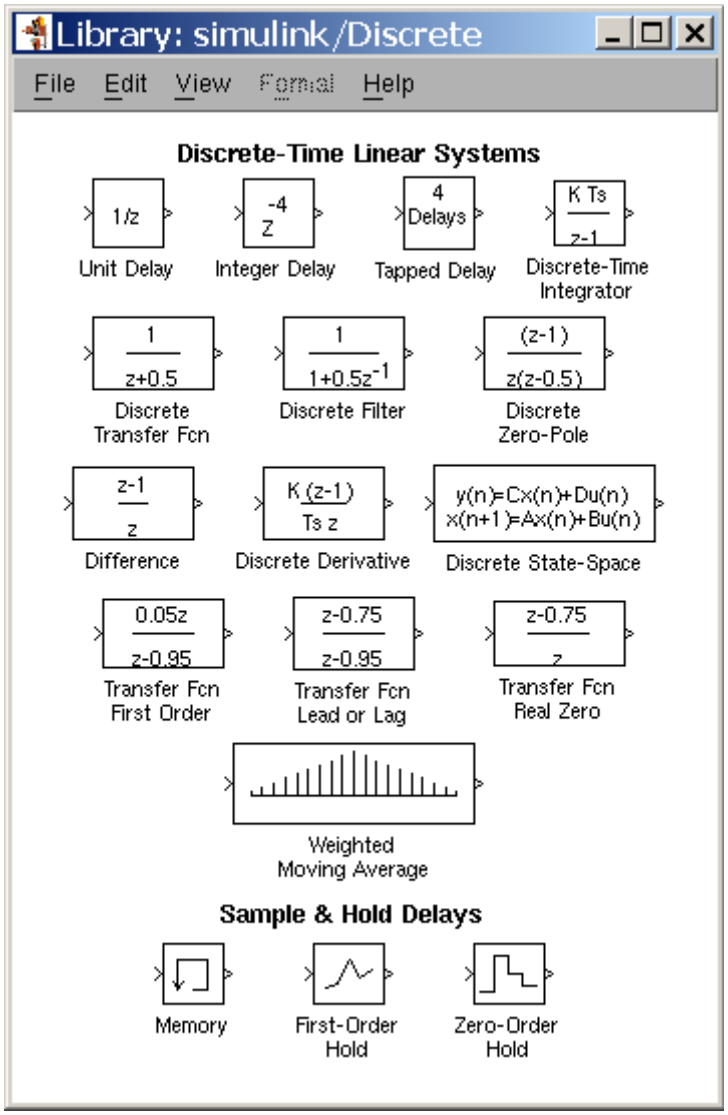

Bild 5 Bibliotheksfenster: Discrete Modellfenster

Ein neues Modellfenster wird über das Menü *File/New/Model* geöffnet.

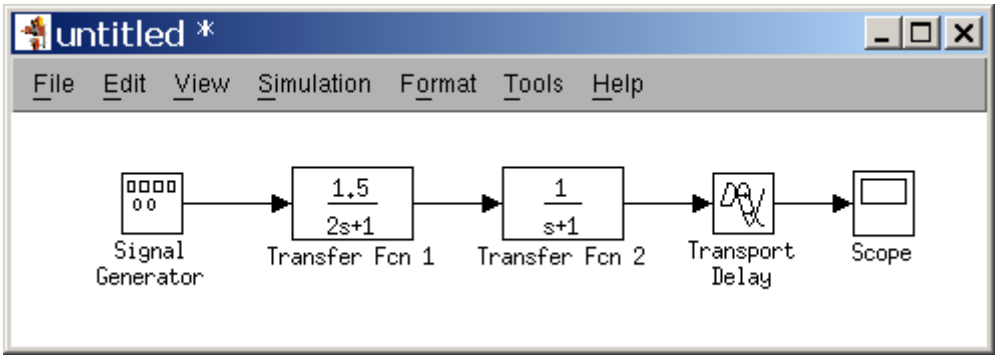

Bild 6 Modellfenster mit einem Beispielsystem

In diesem Fenster findet die Modellierung eines Systems statt. Dabei werden die Bestandteile dem bei Bedarf zu öffnenden Bibliotheksfenster entnommen und miteinander verbunden. Bild 6 zeigt ein klei-

nes System (bestehend aus einem Signalgenerator, zwei nacheinandergeschalteten T1-Übertragungsglieder, einem Totzeitglied und einem Oszilloskop), das noch nicht abgespeichert wurde.

Graphik Editor

Als letztes ist auf den Graphik-Editor hinzuweisen. Er beinhaltet letztlich die Möglichkeit, mit der Maus eine Reihe von graphischen Operationen vorzunehmen. Eine wichtige Rolle spielt dabei das Menü *Edit* der Menüleiste des Modellfensters.

Es enthält die Menüpunkte *Cut*, *Copy*, *Paste*, *Clear* und „*Select All*“. Diese Punkte müssen hier nicht weiter erläutert werden, da sie auch Bestandteil vieler anderer Programme mit einem Graphik-Editor sind. Wesentlich in diesem Zusammenhang ist, dass mit der Maus Graphik-Symbole auch von einem Fenster ins andere verschoben werden können und dass die Symbole durch Linien zu Systemen miteinander verbunden werden können. Hinter jedem Graphik-Symbol steht ein MATLAB-Programm; der Verbindung der Symbole durch Linien entspricht automatisch eine Variablenübergabe zwischen den MATLAB-Programmen.

Starten von Simulink Systemen

Als Voraussetzung muss ein System in einer Datei mit der Endung ****.mdl*** abgespeichert vorliegen. Dieses System kann dann auf verschiedene Arten geöffnet werden. Wenn sich die Datei im Pfad befindet, kann es durch Eingabe seines Namens im „MATLAB Command Window“ geöffnet werden. Weiterhin kann über den Menü-Punkt *File/Open* sowohl von der integrierten Entwicklungsumgebung wie auch von jedem mit einem Menü versehenen Modell jedes beliebige Modell geöffnet werden.

Start der Simulation

Ist die Modellierung eines Systems im Modellfenster abgeschlossen, so liefert das Menü *Simulation* Festlegungen für die Durchführung der Simulation. Hier sind es hauptsächlich die Menüpunkte *Start*, *Stop und „Simulation paramters...“*, die dabei immer wieder benötigt werden. Vor dem Start sollten die Simulationsparameter überprüft werden.

Systemmodellierung

Grundlage der Modellierung eines Systems der analogen oder digitalen Signalverarbeitung sind die in Bibliotheken (Library) zusammengefassten Graphik-Symbole, die Elemente, Bausteine, Teilsysteme oder ganze Systeme darstellen. Auf diesen Bibliotheken beruht zu einem großen Teil die Leistungsfähigkeit des Programms Simulink. Zudem besteht wie bei MATLAB die Möglichkeit, diese Bibliotheken durch eigene Systeme zu ergänzen.

Block Library

Die im Simulink-Bibliotheksfenster (vgl. Bild 4) verfügbaren Bibliothekssymbole bieten den Zugriff auf einen großen Vorrat an weiteren Untergruppen und Blöcken. Systeme der Signalverarbeitung haben einen hierarchischen Aufbau.
Diesen Systemaufbau zeigt Bild 7.

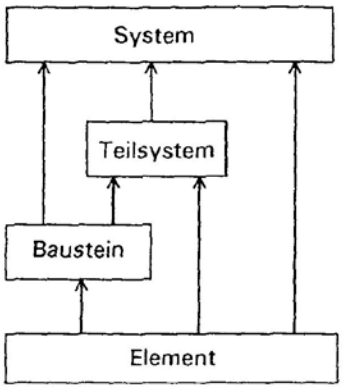

Bild 7 Hierarchischer Aufbau von Systemen

Tabelle 1 Häufig verwendete Symbole der „Simulink Block Library“

Gruppe	Symbol	Ebene	Beschreibung
Signal Sources	Clock	Element	Simulationsschrittakt
	Constant	Element	konstante Folge
	Step Input	Element	Sprungfolge e(n)
Signal Sinks	Graph	System	Oszilloskop
	XY-Graph	System	XY-Oszilloskop
	To Workspace	Variable	Speicherung als Vektor
Discrete Systems	Unit Delay	Element	Verzögerung um 1 Schritt
	Filter	Baustein	Digitalfilter
Linear Systems	Sum	Element	Additionsglied
	Gain	Element	Koeffizientenglied
Nonlinear Systems	Product	Element	Multiplizierer
	Saturation	Baustein	Sättigungskennlinie
	Quantizer	Baustein	Amplitudenquantisierung
	Lookup Table	Baustein	Tabelle(Speicher)
	Transport Delay	Element	Verzögerung (analog)
	Function	Baustein	Block mit skalarer Function
	MATLAB Function	Teilsystem	Block mit Matrix-Function
Connections	Mux	Baustein	mehrere Skalare in Matrix
	Demux	Baustein	Umkehrung von Mux

Die Graphik-Symbole der Simulink-Blöcke gehören verschiedenen Hierarchieebenen an. So gibt es Blöcke, die Elemente sind, wie Addierer, Multiplizierer und Konstantenglieder. Aber auch ganze Bausteine wie z. B. Filter sind als Blöcke vorhanden. Tabelle 1 gibt eine Übersicht über die am häufigsten verwendeten Symbole.

Diese Block Library kann natürlich beliebig ergänzt werden. Dazu besitzt Simulink neben Schnittstellen zu MATLAB eine komfortable Nutzerschnittstelle, auf die im weiteren eingegangen werden soll.

Vorgehensweise:

Bei der Modellierung eines Systems können verschiedene Phasen unterschieden werden:

1 Öffnen eines Modellfensters über das Menü *File/New/Model*
2 Zusammenstellung der Elemente, Bausteine und Systeme mit den Bibliotheks-Symbolen
3 Verbinden der Systemteile an den Schnittstellen
4 Hierarchiebildung durch Gruppieren von Systemteilen
5 Wertzuweisung der Systemparameter

Die Hierarchiebildung und die Wertzuweisung der Systemparameter wird in weiteren Abschnitten ausführlicher beschrieben.

Scope-Block

Der Block ***Scope*** aus der Bibliothek ***Sinks*** (vgl. Bild 4) kann universell zur Anzeige beliebiger Signale verwendet werden.

Bild 8 Block-Symbol Scope

Ein Doppelklick auf das Symbol öffnet das Scope-Fenster aus Bild 9.

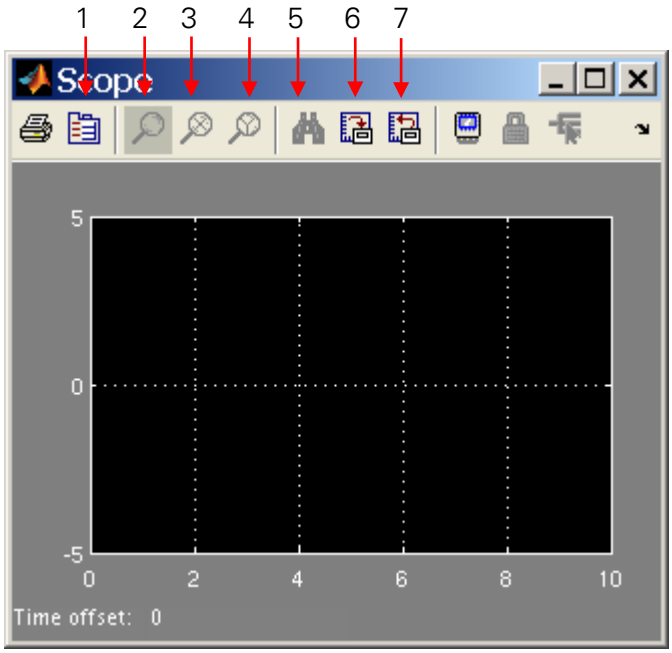

Bild 9 Scope-Fenster

Neben den Zoom-Einstellungen (vgl. Kennzeichnungen 2-4) kann innerhalb der Zeichenfläche ein beliebiger Bereich mit der linken Maustaste aufgezogen und vergrößert werden. Das Betätigen der Schaltfläche „Fernglas“ (vgl. Kennzeichnung 5) liefert eine optimale Skalierung des angezeigten Signals innerhalb der vorgegebenen Grenzen. Weiterhin kann man durch Klicken mit der rechten Maustaste in einen Skalenbereich die Grenzen beliebig einstellen. Die Taste mit der Kennzeichnung 6 speichert die aktiven Einstellungen, so dass die Anzeigen für weitere Experimente gültig bleiben und diese beim Starten des Modells nicht mehr automatisch skaliert werden. Die Taste mit der Kennzeichnung 7 ladet die früher gespeicherten Einstellungen.

Der Block ***Scope*** bietet eine Reihe weiterer Einstellmöglichkeiten, die über die mit 1 gekennzeichnete Schaltfläche aufgerufen werden können. Auf einige wichtige Einstellungen wird nachfolgend näher eingegangen.

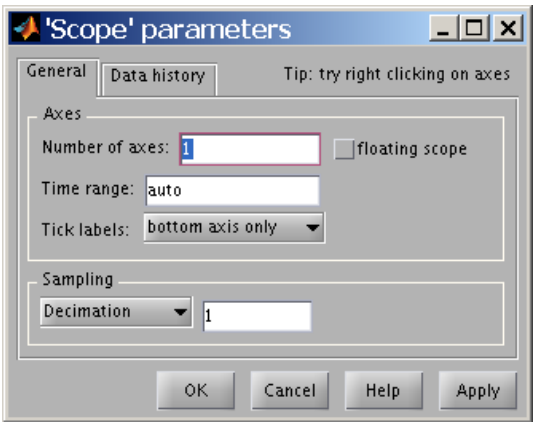

Bild 10 Scope-Einstellungen „General“

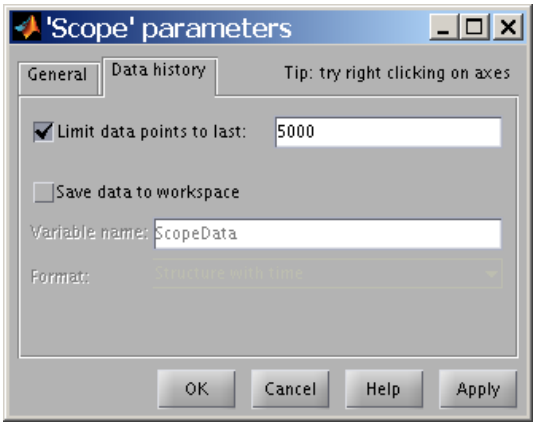

Bild 11 Scope-Einstellungen „Data histoty“

Der Scope-Block verfügt in seiner Grundeinstellung über einen „Inport“ (vgl. Bild 8), der auch Vektorsignale verarbeiten kann (s. später). Schaltet man die Eigenschaft „***floating scope***“ an (vgl. Bild 10), so verschwindet der Eingang des Scope-Blocks. Der Scope-Block stellt dann das Signal der **aktuell markierten** Verbindung dar. Dabei können auch mehrere Verbindungen markiert und angezeigt werden.

Bei längeren Aufzeichnungen kann es sinnvoll sein, die Einstellung „***Limit data points to last:***“ (vgl. Bild 11) auszuschalten bzw. zu erhöhen, um eine längere kontinuierliche Anzeige der Werte zu erreichen.

Zur Veranschaulichung mehrerer Signale bei gleichem Maßstab sei abschließend auf die Möglichkeit verwiesen, ein Vektorsignal mittels Scope-Block darzustellen.

Im Modell aus Bild 12 wird ein Vektorsignal mittels des Simulink-Blockes ***Mux*** aus der Bibliothek „***Signals & Systems***“ gebildet (senkrechter Balken). Der Scope-Block stellt dann alle Signale mit alternierenden Farben in gleichem Maßstab dar.

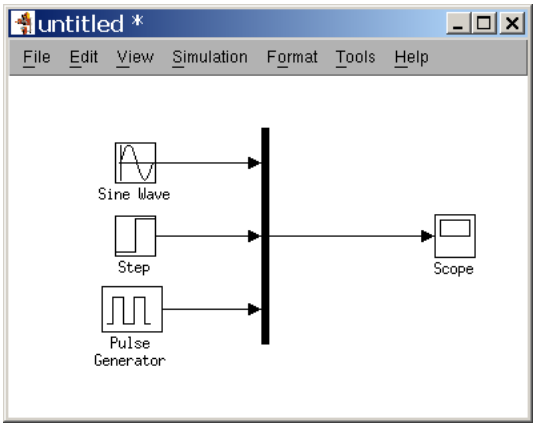

Bild 12 Erzeugung eines Vektorsignals

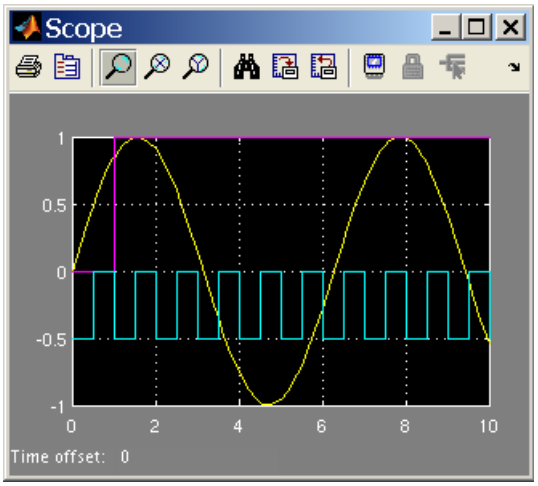

Bild 13 Darstellung eines Vektorsignals

ZUSAMMENWIRKEN VON SIMULINK MIT MATLAB

Die Möglichkeit der Simulation eines erstellten Modells beruht auf der Verbindung der Graphik-Symbole mit MATLAB. Wichtig ist, dass der Benutzer selbst definierte Graphik-Symbole erstellen kann, in welche MATLAB-Anweisungen oder „MATLAB Function Files„eingebaut werden können. In Simulink gibt es 4 verschiedene Möglichkeiten, das mathematische Verhalten von selbst erstellten Symbolen durch MATLAB-Anweisungen festzulegen:

1) durch Verwendung eines „**Function Blocks**“,
2) durch Verwendung eines „**MATLAB Function Blocks**“ ,
3) durch Editieren einer „**S-Function**“,
4) durch Beschreibung eines „**Initialization Commands**“ in einem maskierten Block.

Function Block „Fcn“

Der Block *Fcn* in der Gruppe “*Functions & Tables*” ermöglicht die Definition von Symbolen mit einem Funktionsverhalten, das durch einen MATLAB-Ausdruck festgelegt wird. Das Ausgangssignal dieses Blockes ist eine selbstdefinierte mathematische Funktion des Eingangssignals. Wenn man den Block durch Doppelklick öffnet, kann man in einer Maske z.
B. eine MATLAB-Funktion

```
2*sin(2*pi*0.05*u)
```

eintragen. Hierin ist die Variable `u` die Eingangsgröße des Blocks *Fcn* und das Ergebnis des Funktionsausdrucks ist die Ausgangsgröße. Schließt man an den Eingang dieses Blocks ein Clock-Symbol an, so wird die Eingangsgröße `u` in jedem Simulationsschritt um ein Zeitinkrement erhöht. Dann erzeugt der Fcn-Block an seinem Ausgang ein Sinussignal mit dem Scheitelwert `2` und der Frequenz

`0,05 Hz`. Der „Function Block“ ist beschränkt auf skalare Eingangsgrößen und bestimmte, festgelegte mathematische Ausdrücke (siehe Hilfe).

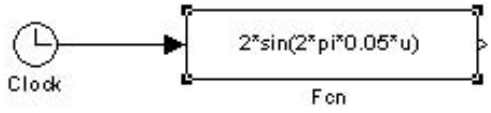

Bild 14 Modellierung eines Sinusgenerators mit einem Fcn-Block

MATLAB Function Block

Dieser Simulink-Block ermöglicht in zweierlei Hinsicht eine weitergehende Anwendung: Es können beliebige selbstgeschriebene „Function Files“ eingesetzt werden und diese können zudem Matrizen als Ein- und Ausgangsgrößen haben.

S-Function Block

Während der „Function Block“ und der „MATLAB Function Block“ MATLAB-Funktionen aufnehmen können, wird in einem „S-Function Block“ eine MATLAB-Systemfunktion eingetragen. Unter Beachtung einer speziellen Syntax kann der Anwender eigene Simulink Blöcke entwerfen und benutzen, indem er ein MATLAB „S-Function File“ entweder in der MATLAB-Programmiersprache oder in C schreibt und den Namen dieser Datei in den „S-Function Block“ einsetzt. Zur Verwendung durch Simulink ist das MATLAB „S-Function File“ zuvor jedoch in Objektcode zu übersetzen (DLL auf Windows-Systemen).

Initialization Command

Durch eine Maskierung von gruppierten Symbolen ergibt sich die Möglichkeit, den Parametern eines Systems oder Teilsystems Werte zuzuweisen. Diese Werte müssen in der Regel zunächst berechnet werden. Dann kann in einer „Initialization Command“ -Zeile der Maske eine MATLAB-Anweisung oder ein „MATLAB Function File“ eingetragen werden, die/das dann beim Starten der Simulation zunächst berechnet werden. Auch hierbei handelt es sich um ein Zusammenwirken von Simulink mit MATLAB. Auf die Möglichkeit der Maskierung von Blöcken und deren Parametrisierung wird später noch einmal detailliert eingegangen.

PARAMETER MODELLIERTER SYSTEME

Bei der Modellierung von Systemen entsteht häufig der Wunsch, den Blöcken ein variierbares Verhalten zu geben. Dies löst man durch Verwendung von Parametern in den entsprechenden Böcken, denen man vor der Simulation spezielle Werte zuweist. Dabei ergeben sich diese Werte in der Regel durch umfangreichere Berechnungen. Betrachtet man z. B. den Block *Filter* aus der Gruppe *Discrete,* so erfordert er die Festlegung der Koeffizienten der Übertragungsfunktion. Die Koeffizienten sind daher Parameter in dem Block. Im folgenden werden 4 Verfahren der Wertzuweisung angeführt.

MATLAB Command Window

Bei diesem Verfahren werden in den Blöcken des modellierten Systems nicht Werte eingetragen, sondern jeweils die Variablen-Namen. Dann können den Variablen im „MATLAB-Command-Window“ Werte zugewiesen werden. Vorteilhaft ist diese Methode, wenn in einer Reihe von Blöcken des modellierten Systems die gleichen Werte erforderlich sind und bei verschiedenen Simulationsläufen geändert werden sollen. Es braucht dann nur in diesen Blöcken der gleiche Variablen-Name eingetragen zu werden.

Script File

Sind in den Blöcken des Systems statt der Werte Variablen-Namen eingetragen und werden die Werte als Ergebnis umfangreicherer Berechnungen erhalten, so ist die Erstellung eines „Script Files“ eine naheliegende Lösung. Beim Starten des „Script Files“ können dann die erforderlichen Eingaben, wie z. B. Grenzfrequenz und Ordnungszahl bei einem Filter, vorgenommen werden. Daraufhin werden die Koeffizienten der Übertragungsfunktion des Filters berechnet und die Werte den entsprechenden Variablen-Namen zugewiesen.

In das „Script File“ kann man schließlich am Ende einen Aufruf desjenigen Simulink-Modells einbauen, mit dem das modellierte System gestartet wird. Mit folgenden, wenigen Programmzeilen werden z. B. die Filterkoeffizienten eines digitalen Filters dimensioniert und danach ein Simulink-Modell *own_system* gestartet. In diesem System findet ein Filter Verwendung, in dem die Koeffizienten als Vektorvariable A(z) und B(z) für das Zähler- und das Nennerpolynom der Übertragungsfunktion H(z) definiert sind.

```
% Script-File für System mit Filter
% und Koeffizientenberechnung

input(...);
[A,B]=butter(..,..);
own_system;
end
```

Auch die mehrmalige Ausführung eines Simulink-Modells, z. B. zur Parameteroptimierung, kann mit einem „Script File“ organisiert werden. Das folgende Beispiel demonstriert durch wiederholte Aufrufe des Modells ***t1_system*** das Verhalten eines T1-Systems.

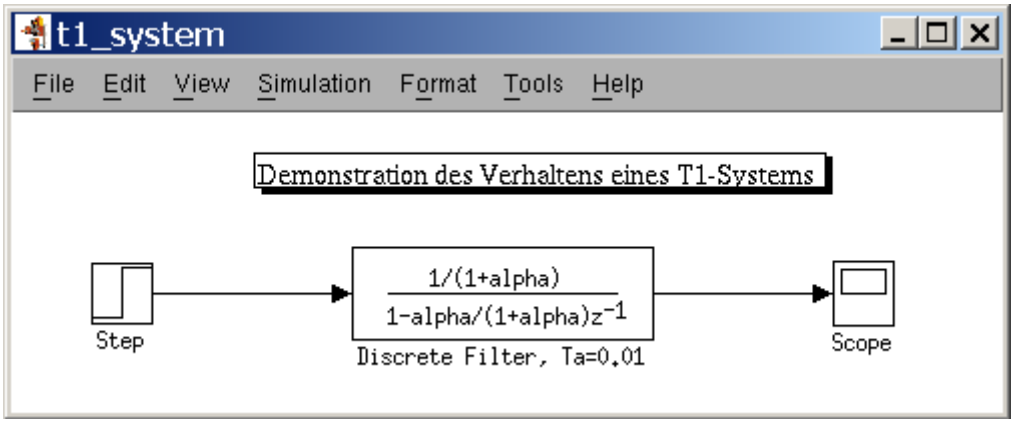

Bild 15 Modell „t1_system“

Dazu wird das Modell ***t1_system*** mit dem „Script File“ ***t1_parametrierung*** aufgerufen.

```
1   function t1_parametrierung()
2
3   % ---Eingabe-Bereich ----------
4   start=1;
5   ende=100;
6   step=1;
7   % ---Ende Eingabe-Bereich -----
8
9   for i=start:step:ende
10  assignin('base','alpha',i);
11  [t,x,y]=sim('t1_system');
12  i
13  pause(0.3);
14  end
```

In den Zeilen 4-6 kann vor der Simulation der Werteumfang des Simulationsparameters ***alpha*** festgelegt werden. Die for-Schleife iteriert über diesen Bereich und startet in Zeile 11 die Simulation des Modells ***t1_system***. Da die Variablen des Scripts nur lokal bekannt sind, muss die Laufvariable ***i*** durch den Befehl in Zeile 10 in den Basis-Workspace ***’base’*** exportiert werden. Dort erhält die lokale Variable ***i*** den Namen ***alpha***. Die Zeile 12 bewirkt die Ausgabe des aktuellen Parameters im „MATLAB Command Window“. In Zusammenhang mit der Verzögerung von ***0.3 s*** aus Zeile 13 kann so der Ablauf direkt im Scope-Fenster verfolgt werden.

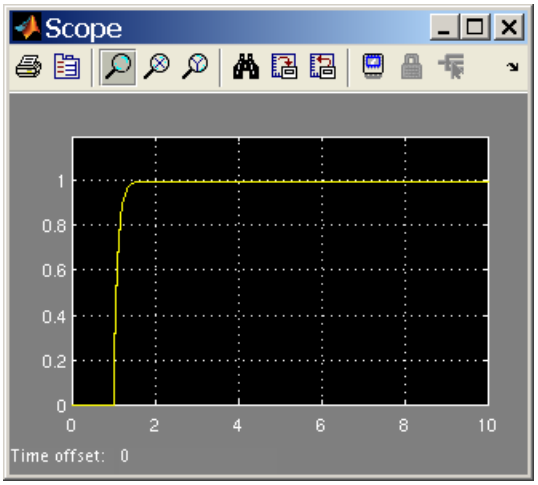

Bild 16 Parameter alpha =10

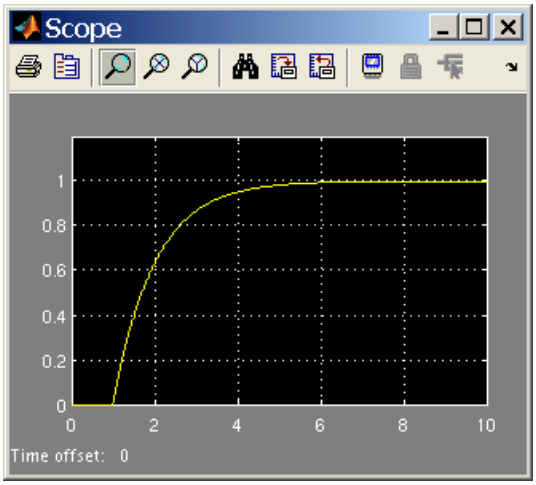

Bild 17 Parameter alpha =100

Konstante

Hier handelt es sich um das direkteste Verfahren der Wertzuweisung. In den Blöcken des modellierten Simulink-Systems werden die erforderlichen Zahlenwerte eingetragen. Gelegentlich ist dies ein unkomplizierter Weg. Ergeben sich diese Werte allerdings durch Berechnungen im „MATLAB Com-

mand Window“, so ist das Verfahren mit einer Variablen günstiger.

Maske

Durch Gruppierung entstandene Systeme oder Teilsysteme können zwecks Wertzuweisung der Systemparameter maskiert werden (s. später). Ein Doppelklick auf das Symbol öffnet dann ein Fenster, in dem Eingangswerte wie z. B. Grenzfrequenz und Ordnungszahl bei einem Filter eingetragen werden. Diese Werte sind die Eingangsgrößen für ein „Function File“, das beim Maskierungsvorgang eingetragen wurde und das dann daraus z. B. die Filterkoeffizienten berechnet.

SYSTEMARTEN

Hier sollen drei Systemarten unterschieden werden:

1. Systeme der digitalen Signalverarbeitung: Der Signaltyp ist hier eine Zahlenfolge, die z. B. in einer Differenzengleichung verarbeitet wird.
2. Systeme der analogen Signalverarbeitung: Der Signaltyp ist hier eine kontinuierliche Zeitfunktion, die z. B. in einer Differentialgleichung verarbeitet wird.
3. Systeme, die sowohl analoge Signale als kontinuierliche Zeitfunktionen als auch Zahlenfolgen enthalten. Hier ist die Digitalisierung ebenfalls Gegenstand der Simulation.

Bei diesen drei Fällen sind die Simulationsparameter unterschiedlich einzustellen. Daher sollen diese, für die Simulation wichtigen Parameter etwas näher beschrieben werden.

Simulationsparameter

Von der Menüleiste des Simulink-Window wird der Punkt *Simulation* näher erläutert. Das Menü des Punktes *Simulation* zeigt u. a. die Unterpunkte:

Start
Stop
Configuration parameters...

Dabei verdient der Punkt “*Configuration parameters...*” besondere Beachtung. Durch seinen Aufruf erscheint das in Bild 18 abgebildete „Simulink Control Panel“.

Unter “*Solver options*” sind die zu Verfügung stehenden numerischen Simulationsverfahren aufgeführt. Für die Simulation ist eines davon auszusuchen. Von den weiteren Simulationsparametern des „Control Panel“ sind besonders wichtig:

Start time, Stop time, Max step size, Min step size

Dabei sollten die Werte *Min Step Size* und *Max Step Size,* die die Simulationsschrittweite festlegen, gleich groß sein, so dass nur noch drei Werte von Bedeutung sind, wobei hinsichtlich der Schrittweite der Begriff „*Step Time*“ verwendet wird. Die Vorbelegung dieser Einstellwerte mit `auto` kann jedoch in den meisten Fällen übernommen werden. Die Einstellung der verbleibenden Parameter hängt von der Systemart und von den Signaltypen ab, die im Modell auftreten. Dies soll in den nachfolgenden

Abschnitten betrachtet werden.

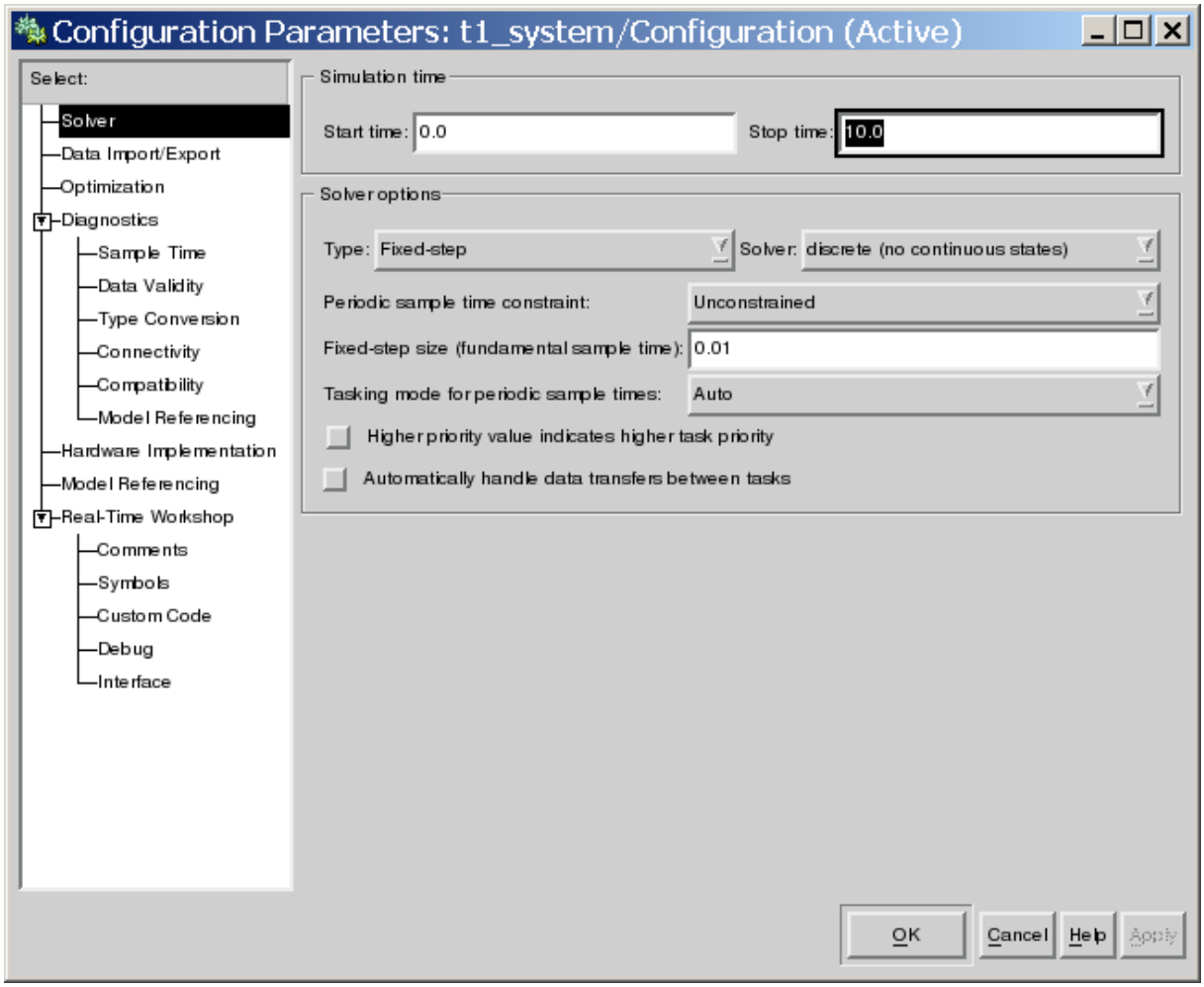

Bild 18 „Simulink control panel“

Digitale Signalverarbeitung

Der Signaltyp ist eine Zahlenfolge `{x(n)}`. An die Stelle der Zeit `t` tritt hier die Folgen-Nummer `n`. Der Parameter „*Start time*“, der den Simulationsbeginn festlegt, wird in der Regel auf Null gesetzt. „*Stop time*“ enthält den Simulationsschritt, bei dem die Simulation beendet wird. Der Block *Clock* erzeugt z. B. eine Zahlenfolge, bei der die Werte mit jedem Simulationsschritt um den Simulationsparameter „*Step time*“ inkrementiert werden. Hat der Parameter „*Step time*“ den Wert 1, so zeigt *Clock* die Folgennummer der Zahlenfolge an.

Analoge Signalverarbeitung

Der Signaltyp ist eine kontinuierliche Zeitfunktion x(t). Die Zeit t wird in Sekunden [s] und die Frequenz in Hertz [Hz] gemessen. Die Eintragungen für „*Start Time*“, „*Stop Time*“ und „*Step Time*“ erfolgen in Sekunden.

Die Simulation kontinuierlicher Signale und Systeme durch numerische Verfahren mit einem Digital-

rechner bedeutet natürlich, dass auch die kontinuierlichen Signale durch Zahlenfolgen dargestellt werden. Die „*Step Time*“ Werte sind dann der zeitliche Abstand zwischen den Stützstellen. Als Richtschnur für die Wahl der „*Step Time*“ kann dienen, dass die spektrale Komponente mit der höchsten Frequenz durch mindestens 10 Stützstellen dargestellt wird. Damit ist die Zeit eine Variable, die sich um ganzzahlige Vielfache der „*Step Time*“ ändert.

Analoge und digitale Signalverarbeitung

Hat man in einem Modellfenster sowohl analoge Signale als auch Zahlenfolgen, so sind drei Fragen zu untersuchen:

1 die Einstellung der „*Step Time*“,
2 die Realisierung der Digitalisierung und
3 die Interpretation des Ablaufs.

Die Fragen sollen an Hand eines Beispiels behandelt werden. Ausgangspunkt sei ein analoges Sinussignal mit einer Frequenz `f0=100 Hz` und einer Periodendauer `T0=l/f0=10 ms`. Zur Darstellung dieses analogen Signals in Simulink werden für eine Periode 100 Stützstellen gewählt. Dann erhält man für den Simulationsparameter „*Step Time*“ einen Wert von 10^{-4}. Zur Erzeugung einer Sinusschwingung mit einer Frequenz von `100 Hz` mit einem „Function Block“ ist in diesem der Ausdruck

```
sin(2*pi*100*u)
```

einzutragen. Für die Modellierung der Digitalisierung bieten sich die Simulink-Blöcke *Unit Delay* als Abtast-Halteglied für die Zeitquantisierung und *Quantizer* für die Amplitudenquantisierung an. Ein Doppelklick auf den Block *Unit Delay* öffnet ein Fenster, in das die Abtastfrequenz als „Sample Time“ eingetragen werden kann. Eine Abtastfrequenz von
z. B. `1 kHz` erfordert dann eine „Sample Time“ von 0,001. Damit entsteht ein Abtast-Haltesignal, das eine Zahlenfolge physikalisch durch eine kontinuierliche Zeitfunktion darstellt (treppenförmiges Signal). Die Simulation des digitalen Systems erfolgt dann statt mit Zahlenfolgen mit Abtast-Haltesignalen.

HIERARCHIEBILDUNG

Das Menü *Edit* der Menüleiste des Modellfensters zeigt u. a. die Unterpunkte „*Create subsystem*”, “*Mask subsystem*” und “*Look under mask*”. Damit verbindet sich die für die Modellierung größerer Systeme so wichtige Hierarchiebildung.

Gruppieren

Der Menüpunkt „*Create subsystem*” ermöglicht eine Gruppenbildung. Eine Gruppe von Symbolen, die zusammen ein Teilsystem bilden, kann durch vorheriges Markieren zu einem neuen Symbol zusammengefasst werden. Dieser Vorgang kann natürlich hierarchisch fortgesetzt werden. Mehrere Gruppensymbole können auch zusammen mit weiteren Einzelsymbolen wiederum ein neues Gruppensymbol bilden.

Maskieren

„*Mask subsystem*" ermöglicht die Maskierung eines Blocks. Das bedeutet die Einrichtung einer Maske, die durch Doppelklicken auf den Block geöffnet wird. In der Maske können Blockparameter, wie z. B. Ordnungszahl und Grenzfrequenz im Falle eines Filters, eingegeben werden. Aus diesen Parametern werden dann z. B. die Koeffizienten des Filters berechnet.
Das Verfahren zur Berechnung der Koeffizienten wird ebenfalls durch die Maskierung eingegeben. Die Einzelheiten zu dem Maskierungsvorgang werden am Beispiel eines Filters erläutert.

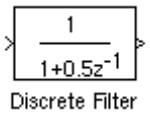

Bild 19 Discrete Filter Block

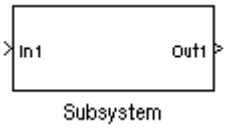

Bild 20 Erzeugtes Subsystem

Dazu ist zunächst der Block „*Discrete Filter*" aus der Bibliothek *Discrete* in ein Modellfenster zu ziehen (Bild 19). Anschließend muss daraus über die Menüpunkte "*Edit/Create subsystem*" ein Subsystem erstellt werden (Bild 20). Simulink bettet den Block „*Discrete Filter*" bei dieser Aktion in einen neuen Block ein, zusammen mit einem „*Inport*"-und einem „*Outport*"- Block, die eine Verbindung zu anderen Blöcken ermöglichen. Ist das Subsystem unmaskiert, kann man durch einen Doppelklick auf das Blocksymbol zum darunterliegenden Modell gelangen (Bild 21).

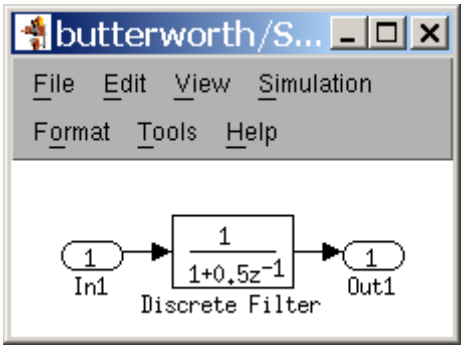

Bild 21 Inhalt des erzeugten Subsystems

Der Block „*Discrete Filter*" ermöglicht die manuelle Eingabe der Koeffizientenvektoren als Variable A(z) und B(z), wie dies im Bild 22 gezeigt ist.

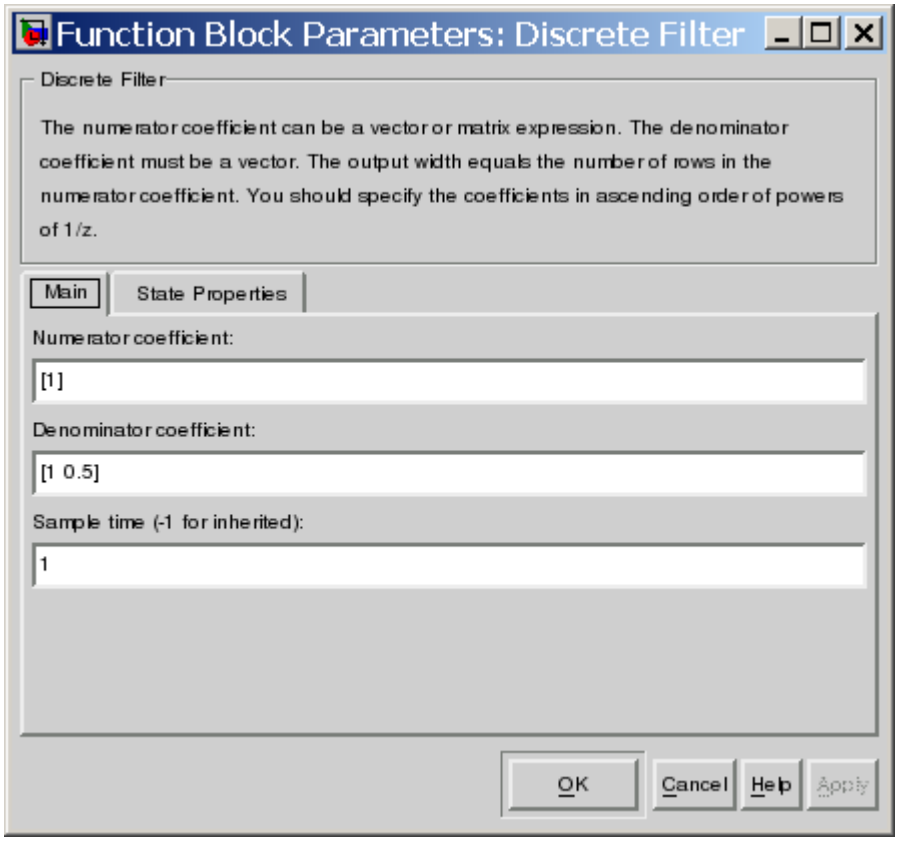

Bild 22 Maskierungsfenster des Blocks “Discrete Filter”

Allerdings müssen diese Koeffizientenvektoren zuvor im „MATLAB Command Window“ errechnet werden, z. B. durch `[A,B]=butter(<Ordnung>,<Grenzfrequenz>);`.
Um dem Nutzer diese Berechnung abzunehmen, kann das Subsystem zu einem maskierten Subsystem gemacht werden, indem es zunächst ausgesucht und anschließend über den Menüpunkt „*Edit/Mask subsystem*” umgewandelt wird.

Trägt man in den „Mask Editor“ (Bild 23) in der Registerkarte “Parameters” z. B. die beiden Edit-Felder “Ordnung” und “Grenzfrequenz” mit den zugehörigen Variablen `ordnung` und `fg` ein, erscheint nun beim Doppelklick auf das Subsystem der folgende Dialog (Bild 25), indem der Nutzer komfortabel die Eingangsbedingungen eines Butterworth-Filters festlegen kann.

Nun müssen noch die Parameter “Numerator” und “Denominator” des diskreten Filterblocks (vgl. Bild 22) errechnet und zugewiesen werden. Die Berechnung erfolgt durch das „Initialization command” `[A,B]=butter(ordnung,fg);` (vgl. Bild 24).

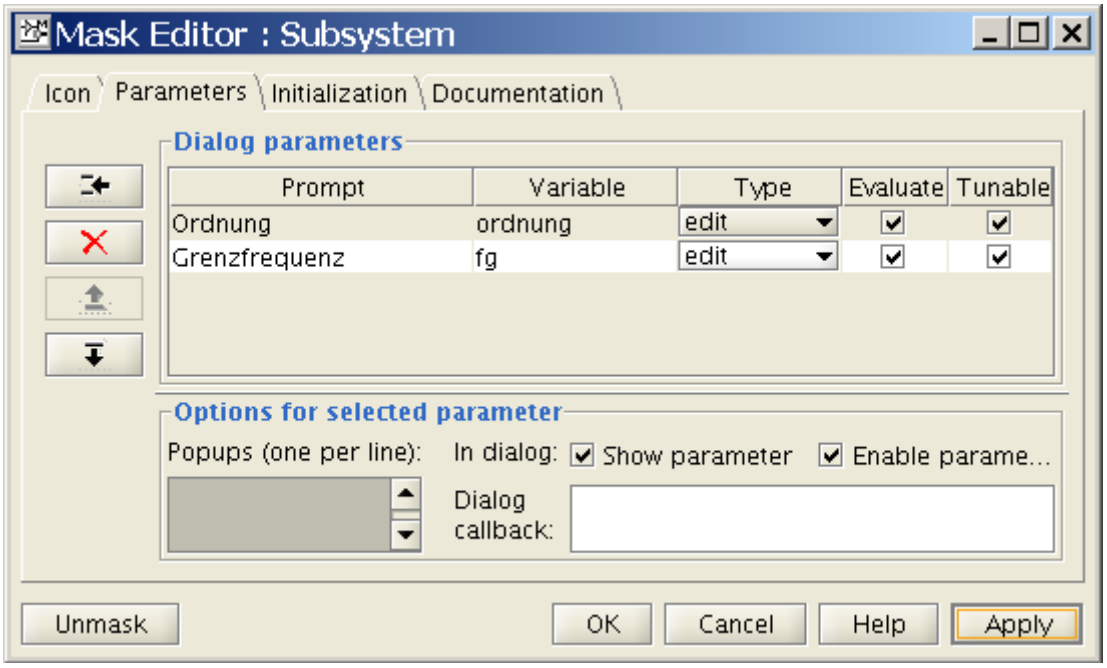

Bild 23 Registerkarte “Parameters” des Maskeditors des Subsystems “Butterworth Filter”

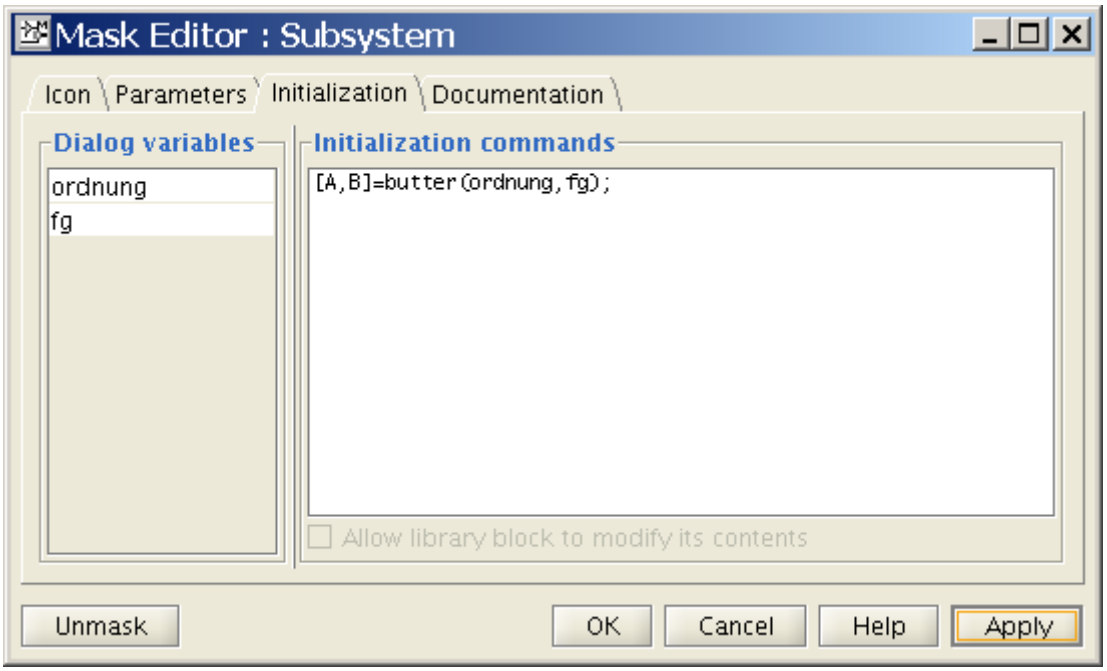

Bild 24 Registerkarte “Initialization” des Maskeditors des Subsystems “Butterworth Filter”

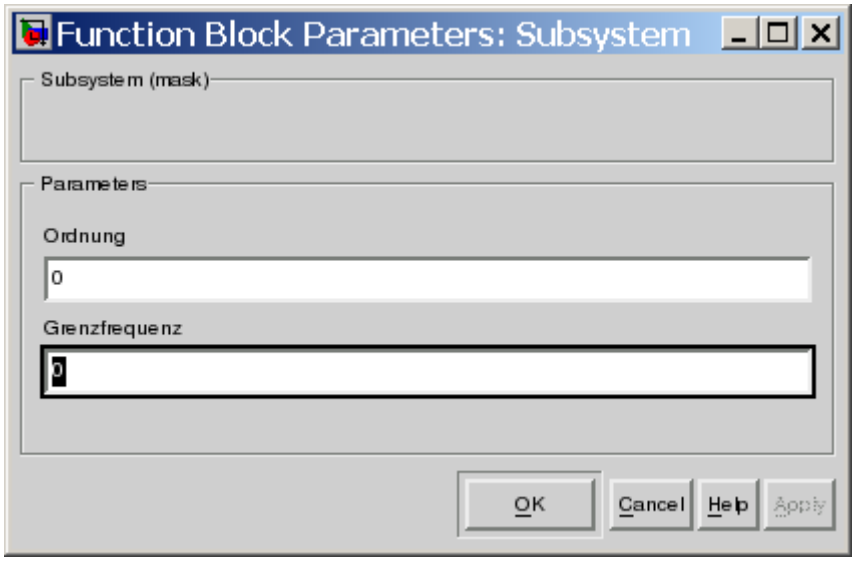

Bild 25 Neu erzeugte “Block Parameter”-Maske des Subsystems “Butterworth Filter”

Vor dem Start eines Simulink-Modells werden alle Initialisierungsroutinen der Blöcke aufgerufen und so in diesem Fall die beiden Koeffizienten A und B der Übertragungsfunktion H(z) des Filterblockes bereitgestellt. Die Zuweisung der beiden Koeffizienten kann nun durch die beiden Einträge “Numerator”: `[A]` und “Denominator”: `[B]` in die entsprechenden Felder des Filterdialogs erfolgen (vgl. Bild 22).

Da nun beim Doppelklick auf das Subsystem die neu erzeugte Maske aufgerufen wird, muss man zum Aufruf des Filterdialogs nun den Weg über den Menüpunkt „*Edit/Look under mask*” gehen, um das darunterliegende Modell anzuzeigen.

Soll die Maskierung von Blöcken wieder aufgehoben werden, so ist dies durch Betätigen von „*Unmask*” im unteren Teil des „Mask Editors“ möglich (vgl. Bild 23 oder Bild 24).