

## Inhalt

1	Das Programmpaket MATLAB.....	2
1.1	Der MATLAB-Interpreter .....	2
1.2	Die wichtigsten MATLAB-Befehle .....	2
1.3	Lösen von linearen Gleichungssystemen .....	7
1.4	Rechnen mit symbolischen Variablen (nur mit Symbolic Toolbox).....	7
1.5	Integrieren (nur mit Symbolic Toolbox) .....	8
1.6	Differenzieren (nur mit Symbolic Toolbox) .....	8
1.7	Rechnen mit komplexen Zahlen.....	9
1.8	Lösen von algebraischen Gleichungen (nur mit Symbolic Toolbox).....	10
1.8.1	Lösen von Gleichungssystemen .....	11
1.9	Lösen von Differentialgleichungen .....	11
1.10	Laplace-Transformation (nur mit Symbolic Toolbox).....	12
1.11	m-Files .....	12
1.11.1	Batch File .....	12
1.11.2	Definition neuer Funktionen .....	13
1.12	Current Directory.....	15
2	Regelungstechnische Anwendungen.....	16
2.1	Analyse linearer Systeme im Frequenzbereich.....	16
2.2	Analyse linearer Systeme im Zeitbereich .....	17
2.3	Weitere Analysefunktionen .....	17
2.4	Transformationen zwischen unterschiedlichen Darstellungsformen .....	18
2.5	Transformationen zwischen zeitkont. und zeitdiskreter Darstellung.....	18
2.6	Zusammenfassung zweier Übertragungsglieder.....	18
2.7	Berechnung der Rückführverstärkungsmatrix für Zustandsregler .....	19
2.8	Berechnung der Rückführverstärkungsmatrix für Luenberg-Beobachter .....	19
3	Das Programmpaket Simulink.....	20
3.1	Beispiel .....	21
3.2	Übersicht Gruppen.....	21
3.3	Übertragungsfunktion.....	22
3.4	Nicht-lineare Kennlinien .....	22
3.5	Ändern der Orientierung eines Blocks .....	23
3.6	Start Simulation.....	23
3.7	Signalverlauf darstellen (Scope) .....	24
3.8	Speichern.....	24
4	Preis und Konkurrenzprodukte.....	25
5	Literatur.....	25


# 1 Das Programmpaket MATLAB

## 1.1 Der MATLAB-Interpreter

MATLAB ist ein Programmpaket für numerische Berechnungen, das insbesondere die Matrizenrechnung unterstützt. Es besteht aus einem MATLAB-Kern, der Algorithmen zur Durchführung wichtiger Matrizenoperationen wie beispielsweise Matrizenmultiplikation oder Eigenwertberechnung umfaßt und außerdem über Funktionen für die grafische Aufbereitung der Ergebnisse verfügt. Erweiterungen in verschiedene Richtungen sind mit zusätzlichen *Toolbox*-en möglich.

Um die hier beschriebenen Übungen durchführen zu können, reicht die in diesem Anhang gegebene Kurzbeschreibung von MATLAB aus. Für weitere Informationen stehen Bücher zur Verfügung (s. Literatur) oder die im Programm installierte Dokumentation. Eine kurze Dokumentation aller Befehle ist auch während der Arbeit mit MATLAB durch den Befehl `help` abrufbar (s.u.).

## 1.2 Die wichtigsten MATLAB-Befehle

**Aufruf:** Das Anklicken des MATLAB-Icons  öffnet ein Fenster, den sogenannten Workspace, in dem man nach dem Prompt „>“ die in MATLAB implementierten Funktionen aufrufen kann.

**Hilfen und Demos:** Alle Befehle haben eine Online-Hilfe, die mit dem Befehl

```
> help <name>
```

aufgerufen wird, wobei für <name> ein bekannter Funktionsname oder der Name einer Toolbox steht. Darüber hinaus können Stichwortsuchen mit dem Befehl:

```
> lookfor <stichwort>
```

durchgeführt werden, wobei <stichwort> ein englischer Suchbegriff ist.

Für eine Übersicht über die Möglichkeiten die MATLAB bietet, gibt es den Befehl

```
> demo
```

Das Programm wird mit

```
> quit
```

beendet.

**Dateneingabe:** MATLAB kennt nur einen Datentyp: Alle Variablen sind Matrizen, deren Elemente Fließkommazahlen doppelter Genauigkeit sind (auch komplex). Die Standard-Darstellung erfolgt mit 5 angezeigten Stellen. Mit

```
> format xyz
```

kann auch eine andere Darstellungsart eingestellt werden.

Skalare sind Matrizen der Dimension (1, 1). Es wird zwischen Zeilenvektoren (1, n) und Spaltenvektoren (n, 1) unterschieden.

Eine Zahl (Skalar) wird folgendermaßen eingegeben:

```
a = 2
```

Ein Zeilenvektor wird folgendermaßen eingegeben:

```
vz = [1 2 3]
```

Ein Spaltenvektor wird folgendermaßen eingegeben:

```
vs = [4; 5; 6]
```

oder auch als transponierter Zeilenvektor:

```
vs2 = vz'
```

Das Produkt zweier Vektoren kann ein Skalar oder eine Matrix sein:

```
M = vs * vz
```

```
s = vz * vs
```

*Anmerkung:*

Skalarprodukt ( $\vec{a} \cdot \vec{b} = |\vec{a}| \cdot |\vec{b}| \cdot \cos \alpha$ ):

```
v_dot = dot(a,b)    (a, b Spaltenvektoren) oder
```

```
v_dot = a'*b
```

Kreuzprodukt ( $\vec{a} \times \vec{b} = |\vec{a}| \cdot |\vec{b}| \cdot \vec{n} \cdot \sin \alpha$ , mit „Rechte-Hand-Regel“):

```
v_cross = cross(a,b)    (a, b Spaltenvektoren)
```

Matrizen werden von eckigen Klammern umrahmt zeilenweise eingegeben. Jede Spalte wird durch einen Leerraum oder ein Komma, jede Zeile durch einen Zeilenvorschub (<Return>-Taste) oder ein Semikolon abgeschlossen.

Die imaginäre Einheit j wird durch ein i oder j dargestellt (Vorsicht: i und j dürfen dann nicht als Variablennamen verwendet werden!).

Beispielsweise kann eine Matrix durch

```
>> A = [2+0.1i    0
        0         2-0.1i];
```

eingegeben werden. Als zweites Beispiel wird die Festlegung einer (4, 3)-Matrix A gezeigt, bei der auch auf einige elementare Funktionen und Konstanten zurückgegriffen wird

```
>> A = [ 2.4    5e-2    5+i    ;    5        -7        8
        pi     log(2)   exp(1);    sqrt(2),sin(3.1),cos(0)]
```

```
>> A =
    2.4000    0.0500    5.0000+1.000i
    5.0000   -7.0000    8.0000
    3.1416    0.6931    2.7183
    1.4142    0.0416    1.0000
```

Für größere Matrizen ist die interaktive Dateneingabe nicht geeignet. Mit einem beliebigen Editor legt man sich deshalb eine Datei an, in die die Elemente der Matrix zeilenweise geschrieben werden, wobei hier nur Zahlen und keine Funktionsaufrufe benutzt werden dürfen. Ist der Name einer solchen Datei z.B. B.dat, so ist nach dem Aufruf des Befehls

```
>> load B.dat
```

die Matrix B im Workspace verfügbar.

Ein Vektor kann auch automatisch erzeugt werden:

`x = 0 : 0.1 : 2` (Startwert : Schrittweite : Endwert):

Dies könnten die x-Werte einer Funktion sein. Die zugehörigen y-Werte ergeben sich z.B. für die Quadratfunktion zu

`y = x .* x`

Der Operator `.*` bedeutet, daß nicht der komplette Vektor `x` mit sich selbst multipliziert wird, sondern jeweils die einzelnen Elemente des Vektors. `y` ist also auch ein Vektor der selben Dimension wie `x`!

Für die Exponentialfunktion schreibt man:

`y2 = exp(x)`

Will man die Funktion grafisch darstellen, geschieht das mit dem Befehl

`plot(x, y)`

Will man mehrere Linien z.B. für einen schwarz-weiß-Druck unterscheidbar machen, findet man eine Anleitung unter

`help plot`

Soll eine Linie massiv und die andere gestrichelt sein, lautet der Befehl

`plot(x, y, '-', x, y2, '--')`

Daneben gibt es auch noch die Möglichkeit der voll- oder halblogarithmischen Darstellung.

**Funktionsaufrufe:** Will man Berechnungen mit MATLAB durchführen, so wird das Ergebnis des Aufrufes der Funktion `befehl` mit der Eingangsvariablen `einvar` einer Ausgabevariablen `ausvar` durch das Gleichheitszeichen zugeordnet:

» `ausvar = befehl(einvar)`

Definiert man keine neue Ausgabevariable., so wird das Ergebnis der letzten Berechnung in der Variable `ans` (für `answer`) gespeichert. Mehrere Eingabevariablen werden durch Kommata getrennt. Besteht das Ergebnis des Funktionsaufrufes aus mehreren Elementen, so sind die Ausgabevariablen in eckige (Matrix)-Klammern zu setzen:

» `[aus1, aus2] = befehl(ein1, ein2, ein3)`

Will man die Bildschirmausgabe der Ergebnisse unterdrücken (was insbesondere bei Zwischenergebnissen in Form großer Matrizen sinnvoll ist), so wird das jeweilige Kommando mit einem Semikolon beendet:

» `ausvar = befehl(einvar);`

Die Dimension einer Matrix A erhält man mit dem Befehl

```
» size(A)
```

Das Ergebnis ist ein Vektor, dessen erstes Element die Zeilenanzahl und dessen zweites die Spaltenanzahl angibt. Für die o.a. Matrix gilt

```
» size(A)
```

```
ans = 4    3
```

**Indizierung:** Auf die Elemente einer Matrix kann einzeln zugegriffen werden. Benötigt man beispielsweise das Element mit dem Index (2, 3) der Matrix A als Variable `element`, so gibt man

```
» element = A(2,3)
```

ein.

**Spezielle Befehle:** Um spezielle Matrizen einfacher anlegen zu können, gibt es den Befehl

```
» I = eye(n);
```

der eine (n, n)-Einheitsmatrix I anlegt und den Befehl

```
» 0 = zeros(n, m);
```

der eine (n, m)-Nullmatrix 0 anlegt.

Eine Matrix A wird transponiert, indem man einen Apostroph anhängt:

```
» At = A';
```

Auf diese Weise ist auch eine Umwandlung von Zeilen- in Spaltenvektoren und umgekehrt möglich.

Die wichtigsten Matrizenfunktionen dienen der Berechnung der Eigenwerte von A (Diagonalelemente von D) und der zugehörigen Eigenvektoren (Spalten in Matrix V)

```
» [V, D] = eig(A),
```

der Berechnung der Inversen

```
» inv(A),
```

der Berechnung der Determinante

```
» det(A),
```

und der Berechnung des Rangs

```
» rank(k)
```

einer Matrix A.

## Weitere Befehle

Polynome werden in MATLAB durch Vektoren dargestellt, die die Polynomkoeffizienten (in Richtung fallender Exponenten) enthalten, beispielsweise wird

$$P(x) = x^4 + 3x^3 + 1$$

durch

`p = [1 3 0 0 1]` dargestellt.

## Die Funktion

» `roots(p)`

liefert die Nullstellen (real oder komplex) des reellen Polynoms  $P(x)$ .

## Die Funktion

» `conv(p1, p2)`

berechnet das Produkt der reellen Polynome  $P_1(x)$  und  $P_2(x)$ .

## Die Funktion

» `[q, r] = deconv(p1, p2)`

berechnet die Polynomdivision  $q = P_1(x) / P_2(x)$  mit dem Rest  $r$ .

## Die Funktionen

» `abs(k)`

bzw.

» `angle(k)`

liefern den Betrag bzw. den Winkel der komplexen Zahl  $k$ .

### 1.3 Lösen von linearen Gleichungssystemen

$$\begin{array}{rrcr} x_2 & + & 3x_3 & -x_4 & = & 1 \\ -2x_1 & & & +x_3 & +x_4 & = & 2 \\ -x_1 & + & 2x_2 & + & 2x_3 & + & x_4 & = & 6 \\ x_1 & & -x_2 & & +x_3 & - & 4x_4 & = & -6 \end{array}$$

in Matrixschreibweise:

$$\begin{pmatrix} 0 & 1 & 3 & -1 \\ -2 & 0 & 1 & 1 \\ -1 & 2 & 2 & 1 \\ 1 & -1 & 1 & -4 \end{pmatrix} \cdot \vec{x} = \begin{pmatrix} 1 \\ 2 \\ 6 \\ -6 \end{pmatrix} \quad \text{oder} \quad A \cdot \vec{x} = \vec{b}$$

$$A \cdot \vec{x} = \vec{b} \quad | A^{-1} \cdot$$

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

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

$$\vec{x} = A^{-1} \cdot \vec{b} \quad \rightarrow \text{kann einfach mit MATLAB gelöst werden:}$$

$A$  und  $b$  belegen  $\rightarrow x = \text{inv}(A) * b$     alternativ:  $x = A \backslash b$

(\: „Matrix-Linksdivision“ – Reihenfolge bei Matrizenoperationen ist wichtig!)

### 1.4 Rechnen mit symbolischen Variablen (nur mit Symbolic Toolbox)

```
>> syms a b                                % Einführung symbolischer Variablen

>> S = (a + b)^2                            % umzuformender allgemeiner Ausdruck
>> simple(S)                               % erzeugt verschiedene Vorschläge

factor: (a+b)^2
expand: a^2+2*a*b+b^2
...
ans = (a+b)^2                               % empfohlene Lösung

oder

>> S = a^2 + 2*a*b + b^2                    % umzuformender allgemeiner Ausdruck
>> simple(S)                               % erzeugt verschiedene Vorschläge

factor: (a+b)^2
expand: a^2+2*a*b+b^2
...
ans = (a+b)^2                               % empfohlene Lösung
```

## 1.5 Integrieren (nur mit Symbolic Toolbox)

$$W = \int_{0m}^{5m} 2 \frac{N}{m} \cdot s \, ds = 2 \frac{N}{m} \cdot \int_{0m}^{5m} s \, ds = 2 \frac{N}{m} \cdot \left[ \frac{s^2}{2} \right]_{0m}^{5m} = \frac{1}{2} \cdot 2 \cdot \frac{N}{m} \cdot (25m^2 - 0m^2) = 25Nm$$

```
>> syms s % Einführen der symbolischen Variablen „s“
>> int(2*s, s, 0, 5) % Integrand, Int-Variable, untere Grenze, obere Grenze
ans = 25
```

oder nur Stammfunktion:

```
>> syms s
>> int(2*s, s) % Integrand, Int-Variable
ans = s^2
```

Anmerkung: Für die untere bzw. obere Grenze kann auch „inf“ ( $\infty$ ) gesetzt werden.

## 1.6 Differenzieren (nur mit Symbolic Toolbox)

- Ableitung  $\frac{d(ax^3+2x+5)}{dt} = 3 \cdot a \cdot x^2 + 2$ :

```
>> syms a x
>> diff(a*x^3+2*x+5, 'x') % Funktion, Variable für Ableitung
ans = 3*a*x^2+2
```

- n-te Ableitung  $\frac{d^2(ax^3+2x+5)}{dt^2} = 6 \cdot a \cdot x$ :

```
>> syms a x
>> diff(a*x^3+2*x+5, 'x', 2)
ans = 6*a*x
```



## 1.7 Rechnen mit komplexen Zahlen

```
>> w = 2*pi*50
      w = 314.1593

>> U = 220;    % ";" unterdrückt das Bildschirm-Echo

>> C = 220e-6;

>> wL = 40;

>> R1 = 20;

>> R2 = 5;

>> Z_L = j*wL
      Z_L = 0 +40.0000i

>> Z_C = 1/(j*w*C)
      Z_C = 0 -14.4686i

>> Z = Z_C + (1/((1/Z_L)+(1/R1))) + R2
      Z = 21.0000 - 6.4686i

>> abs(Z)
      ans = 21.9737

>> angle(Z)*180/pi
      ans = -17.1204

>> I = U/Z
      I = 9.5683 + 2.9473i

>> abs(I)
      ans = 10.0120

>> angle(I)*180/pi
      ans = 17.1204
```

## 1.8 Lösen von algebraischen Gleichungen (nur mit Symbolic Toolbox)

(nicht nur Polynome...) Wenn keine analytische Lösung gefunden werden kann, wird eine numerische Näherungslösung bestimmt.

$$x \cdot 2 = 6$$

```
% ohne Einführung symbolischer Variablen!  
>> solve('x * 2 = 6', 'x')
```

```
ans = 3
```

oder  $a \cdot \sin(y) = 2$

```
>> solve('a * sin(y) = 2') % Löst nach der Variablen auf,  
                           % die x alphabetisch am nächsten  
                           % steht
```

```
ans = asin(2/a)
```

oder allgemeine Lösung von  $ax^2 + bx + c = 0$

```
>> solve('a*x^2 + b*x + c = 0')
```

```
ans =  
-1/2*(b-(b^2-4*a*c)^(1/2))/a  
-1/2*(b+(b^2-4*a*c)^(1/2))/a
```

### 1.8.1 Lösen von Gleichungssystemen

```
>> [x1, x2] = solve('2*x1^2 + 5*x2 = 12', 'x1 - 3*x2 = -5', 'x1', 'x2')
% Angabe von 'x1', 'x2' wäre nicht nötig gewesen...
% Ergebnis in lexikographischer Ordnung: [x1,x2] oder [a,b,c]
x1 =
[ -11/6]
[      1]

x2 =
[ 19/18]
[      2]
```

D. h. es gibt zwei Lösungen (quadratische Gleichung):

Lösung 1:  $x_1 = -11/6$ ;  $x_2 = 19/18$

Lösung 2:  $x_1 = 1$ ;  $x_2 = 2$

**Alternative:** Verwendung von „fsolve“. Hier wird zur Lösung die Methode der kleinsten Quadrate verwendet. D.h. es wird keine analytische sondern numerisch eine (Näherungs-) Lösung bestimmt. Dazu müssen Startwerte vorgegeben werden, die der erwarteten Lösung nahe sein sollten, um zu vermeiden, dass man in die „falsche Richtung läuft“.

Beispiel für Problem „ $x \cdot 2 = 6$ “:

```
fsolve('x * 2 - 6', 1)
```

Parameter: Gleichung in der Form:  $f(x) = 0$ ; Startwert für  $x$

Vorteil: Kann auch noch Lösungen liefern, wenn „solve“ versagt, benötigt keine „Symbolic Toolbox“.

### 1.9 Lösen von Differentialgleichungen (nur mit Symbolic Toolbox)

$R \cdot C_{ges, Reihe} \cdot \dot{i}_R(t) + i_R(t) = 0$  mit  $i_R(0) = 0,48 A \Rightarrow$

```
>> dsolve('R*C*DI + I = 0, I(0) = 0.48')
% "D" steht für die Ableitung; D2 für d^2/dt^2
```

```
ans = 12/25*exp(-1/R/C*t)
```

Eventuell plotten (mit  $R \cdot C = 0,5 s$ ):

```
>> t = 0:0.1:3 % von 0 bis 3 Sekunden
```

```
>> I = 12/25*exp(-t/0.5)
>> plot(t, I)
```

## 1.10 Laplace-Transformation (nur mit Symbolic Toolbox)

Laplace Transformation: `laplace()`

Laplace Rücktransformation: `ilaplace()`

Beispiele:

```
>> syms s t w

>> laplace(5*exp(-2*t))
ans = 5/(s+2)

>> laplace(sin(w*t))
ans = w/(s^2+w^2)

>> ilaplace(1/(s^2 + 4))
ans = 1/2*sin(2*t)
```

## 1.11 m-Files

Mit einem Texteditor (z.B. auch von Matlab zur Verfügung gestellt) können sogenannte **m-files** erstellt werden, die mehrere Befehle hintereinander ausführen oder auch neue Funktionen definieren. Die Endung des Dateinamens muß dabei „.m“ sein. Der Aufruf erfolgt durch Eingabe des Dateinamens ohne „.m“.

Siehe auch Kap. 1.12.

### 1.11.1 Batch File

```
% test_batch.m
%
% Claus Brüdigam, 30.4.2009

t = 0:1:9
y = [0, 1, 4, 9, 15, 20, 21, 18, 10, 2]

plot(t, y)
```

### 1.11.2 Definition neuer Funktionen

Zum Beispiel enthält das File `demo_fct.m` den folgenden Inhalt:

```
function [c_abs, c_rad, c_deg] = demo_fct(c_number)

% demo_fct calculates magnitude and angle of a complex number
%
% Claus Brüdigam
% 18.01.2009

c_abs = abs(c_number);
c_rad = angle(c_number);
c_deg = angle(c_number)*180/pi;
```

Wenn sich dieses File im „Current Directory“ (s. Kap. 1.12) oder im Matlab-Path (Anzeigen mit „>> path“) befindet, erhält man im Command Window das folgende Ergebnis:

```
>> [answ1, answ2, answ3] = demo_fct(1+1j)
answ1 =    1.4142
answ2 =    0.7854
answ3 =    45
```

## Weiteres Beispiel: Berechnung der T-Ersatzschaltung (Vierpol)

```
function [Za, Zb, Zc] = z2t(Z)

% [Za, Zb, Zc] = Z2T(Z) calculates T-equivalent circuit from
% given Z-matrix
%
%      ---[Za]-----[Zc]---
%              |
%              |
%      [-----]
%      |   Zb   |
%      |-----|
%              |
%
% Claus Brüdigam
% 23.04.2009

format short g

% Fehler abfangen:
if Z(1,2) ~= Z(2,1),
    error('Error in matrix elements: Z(1,2) ~= Z(2,1)');
end

Za = Z(1,1) - Z(1,2);
Zb = Z(1,2);
Zc = Z(2,2) - Z(1,2);
```

im Command Window:

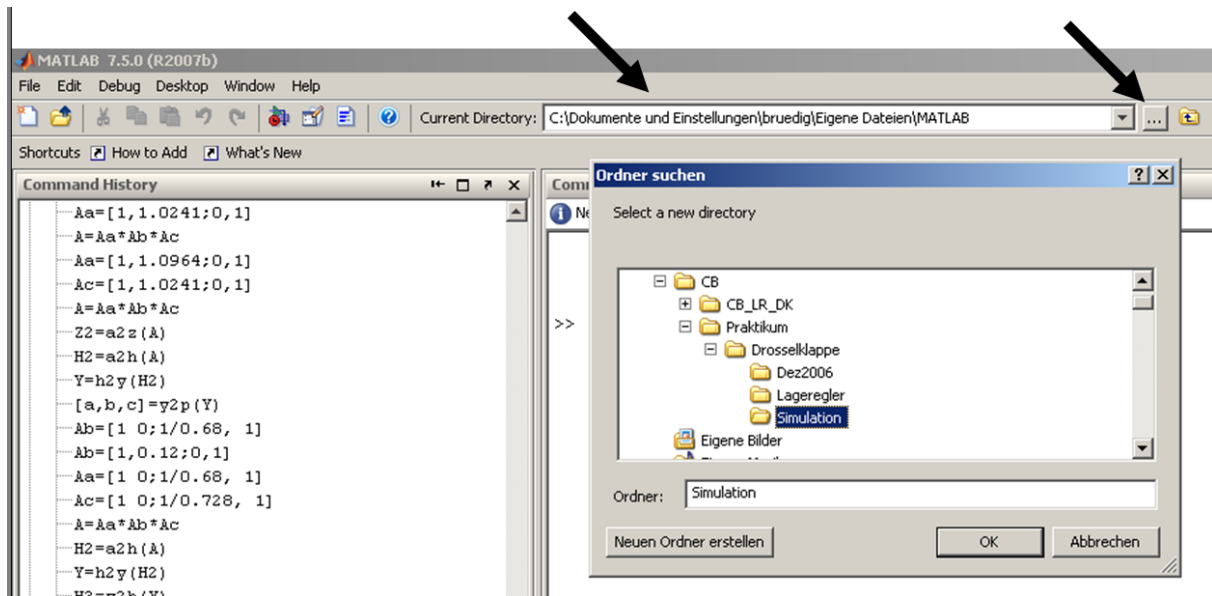
```
>> Z=[3,2;2,4]
Z =
     3     2
     2     4

>> [Ra, Rb, Rc] = z2t(Z)

Ra = 1
Rb = 2
Rc = 2
```

## 1.12 Current Directory

Wird in der Menüleiste angezeigt und kann beliebig geändert werden:



Hier sucht Matlab nach Daten bzw. Funktionen (zusätzlich zum MATLABPATH, der mit „>> path“ angezeigt bzw. geändert werden kann).

## 2 Regelungstechnische Anwendungen

Die Befehlsübersicht über die regelungstechnischen Anwendungen der *Control System Toolbox* erhält man mit

```
» help control
```

Regelungstechnische Demonstrationsprogramme sind verfügbar unter

```
» demo
```

 und dann

+ MATLAB

- Toolboxes

Control System

### 2.1 Analyse linearer Systeme im Frequenzbereich

Kontinuierliche Systeme werden durch die gebrochen rationale Übertragungsfunktion  $F(s)$  beschrieben, die in Zähler- und Nennerpolynom zerlegt werden kann:

$$F(s) = \frac{Z(s)}{N(s)}$$

Dementsprechend wird  $F(s)$  in MATLAB durch ein Paar  $(z, n)$  – Vektordarstellung des Zähler- bzw. Nennerpolynoms – erzeugt:

```
» z = [2] % Zähler: 2
» n = [3, 0, 4] % Nenner: 3s² + 4
» F = tf(z, n) % F ist damit eine Übertragungsfunktion
» [z, n] = tfdata(F, 'v') % Ergibt Zähler u. Nenner von F(s)
```

#### **Alternative:**

```
» s = tf([1, 0], [1]) % Def. komplexe Variable s
» F = 1/(2*s + 1) % F ist damit eine Übertragungsfunktion
```

#### **Ergänzung:**

```
» F = tf(z, n, 'InputDelay', 2.0) % zus. 2 s Totzeit
```

**Anmerkung:** je nach Matlab-Version ist die Übertragungsfunktion als Funktionsargument entweder in der Form  $(z, n)$  oder  $(Fkt)$  einzugeben.

Z.B.: `printsys(z, n)` oder `printsys(F)`

(teilweise ist auch nur eine Variante zulässig – z.B. `[A,B,C,D] = tf2ss(z,n)`)

Folgende Funktionen werden häufig benötigt:

```
» printsys(F) % Ausgabe der Übertragungsfunktion auf dem Bild-
```



- » `printsys(z, n)` schirm
- » `dcgain(z, n)` Berechnung der statischen Verstärkung  $k_s = F(0)$

Da Totzeitsysteme keine gebrochen rationale Übertragungsfunktion haben, können diese Systeme nicht ohne eigenen Programmieraufwand mit MATLAB behandelt werden. Man kann jedoch mit der Funktion

- » `[z, n] = pade(Tt, n)`

die Padé-Approximation n-ter Ordnung berechnen, wenn der Variablen `Tt` zuvor der Wert der Totzeit zugewiesen wurde. Um keine numerischen Probleme zu bekommen, wählt man `n` nicht größer als 5.

## 2.2 Analyse linearer Systeme im Zeitbereich

Das Zustandsraummodell

$$\begin{aligned}\dot{\vec{x}}(t) &= A \cdot \vec{x}(t) + B \cdot \vec{u}(t); & x(0) &= x_0 \\ \vec{y}(t) &= C \cdot \vec{x}(t) + D \cdot \vec{u}(t)\end{aligned}$$

wird durch die Systemmatrix  $A$ , die Steuermatrix  $B$ , die Beobachtungsmatrix  $C$  und die Durchgangsmatrix  $D$  dargestellt. Dabei ist  $\vec{x}(t)$  der Zustandsvektor,  $\vec{u}(t)$  der Eingangsvektor und  $\vec{y}(t)$  der Ausgangsvektor.

- » `printsys(A, B, C, D)` Ausgabe der Matrizen des Zustandsraummodells auf dem Bildschirm
- » `dcgain(A, B, C, D)` Berechnung der statischen Verstärkung  $k_s = -C \cdot A^{-1} \cdot B + D$
- » `eig(A)` Berechnung der Eigenwerte der Matrix (Pole des Systems)

## 2.3 Weitere Analysefunktionen

Die folgenden Funktionen sind für Zeitbereichs- und Frequenzbereichsmodelle sehr ähnlich:

- » `pzmap(F)` Grafische Darstellung der Pol-/Nullstellen
- » `pzmap(z, n)`
- » `pzmap(A, B, C, D)`
- » `step(F)` Berechnung der Übergangsfunktion und grafische Ausgabe auf dem Bildschirm
- » `step(z, n)`
- » `step(A, B, C, D)`
- » `impulse(F)` Berechnung der Gewichtsfunktion und grafische Ausgabe auf dem Bildschirm
- » `impulse(z, n)`
- » `impulse(A, B, C, D)`
- » `bode(F)` Berechnung des Bode-Diagramms und grafische Darstellung auf dem Bildschirm
- » `nyquist(F)` Berechnung der Ortskurve und grafische Darstellung auf dem Bildschirm
- » `[km, pm, wk, wp] = margin(F)` Berechnung des Amplitudenrandes  $k_m$  bei  $\omega_k$

- » `rlocus(F)` und des Phasenrandes  $\varphi_m$  bei  $\omega_\varphi$  zeichnet die Wurzelortskurve (Verstärkung für einen beliebigen Punkt durch Mausklick)
- » `rlocfind(F)` berechnet zu einem beliebigen Punkt der gezeichneten WOK die zugehörige Verstärkung und zeigt alle anderen Pole bei dieser Verstärkung an.

## 2.4 Transformationen zwischen unterschiedlichen Darstellungsformen

Die folgenden Funktionen dienen der Überführung eines Zustandsraummodells mit verschwindendem Anfangszustand  $x_0 = 0$  in eine Übertragungsfunktion oder umgekehrt bzw. der Überführung des Zustandsraummodells in eine kanonische Normalform.

- » `[z, n] = ss2tf(A, B, C, D)` Berechnung der Übertragungsfunktion  
$$F(s) = C(s\mathbf{I} - A)^{-1} \cdot B + D$$
aus dem Zustandsraummodell
- » `[A, B, C, D] = tf2ss(z, n)` Berechnung des Zustandsraummodells in Regelungsnormalform, das die gegebene Übertragungsfunktion besitzt

## 2.5 Transformationen zwischen zeitkont. und zeitdiskreter Darstellung

- » `F_d = c2d(F_c, Ts, 'method')`  
Berechnung der z-Übertragungsfunktion (zeitdiskret) aus der s-Übertragungsfunktion (zeitkontinuierlich). *Ts*: Abtastzeit; method: z.B. Tustin

## 2.6 Zusammenfassung zweier Übertragungsglieder

Für die drei Standardfälle einer Zusammenschaltung zweier Übertragungsglieder gibt es Funktionen, die aus den Übertragungsfunktionen bzw. den Zustandsraummodellen der beiden Elemente das Modell der Zusammenschaltung zu berechnen gestatten.

Reihenschaltung:

- » `F = series(F1, F2)` % auch als `F1 * F2` möglich
- » `[A, B, C, D] = series(A1, B1, C1, D1, A2, B2, C2, D2)`

Parallelschaltung:

- » `F = parallel(F1, F2)` % auch als `F1 + F2` möglich
- » `[A, B, C, D] = parallel(A1, B1, C1, D1, A2, B2, C2, D2)`

Rückführschaltung (Kreisstruktur):

- » `F = feedback(F1, F2)` % neg. feedback!
- » `F = feedback(F1, F2, +1)` % pos. feedback
- » `[A, B, C, D] = feedback(A1, B1, C1, D1, A2, B2, C2, D2)`

## 2.7 Berechnung der Rückführverstärkungsmatrix für Zustandsregler

Bei gegebenen Matrizen  $A$  und  $B$  und gewünschten Polen  $P = [p_1, p_2, \dots, p_n]$  kann die Rückführverstärkungsmatrix  $K$  für eine vollständige Zustandsrückführung (Zustandsregler) berechnet werden:

» `K = place(A, B, P)`

**Achtung:** Der Algorithmus der Funktion „place()“ erlaubt maximal so viele gleiche Pole wie Eingangssignale vorhanden sind.



## 2.8 Berechnung der Rückführverstärkungsmatrix für Luenberg-Beobachter

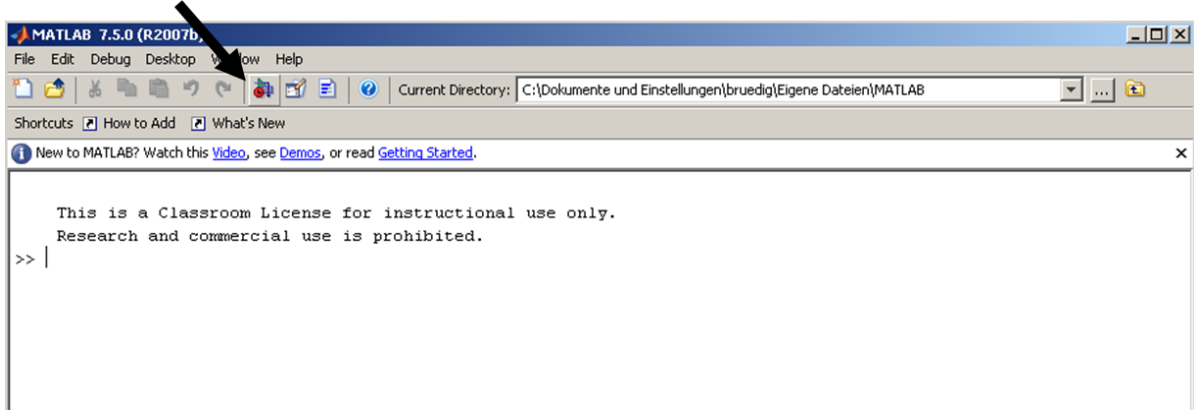
Bei gegebenen Matrizen  $A$  und  $C$  und gewünschten Polen  $P = [p_1, p_2, \dots, p_n]$  kann die Rückführverstärkungsmatrix  $L$  für den Beobachter berechnet werden:

» `L = place(A', C', P)`

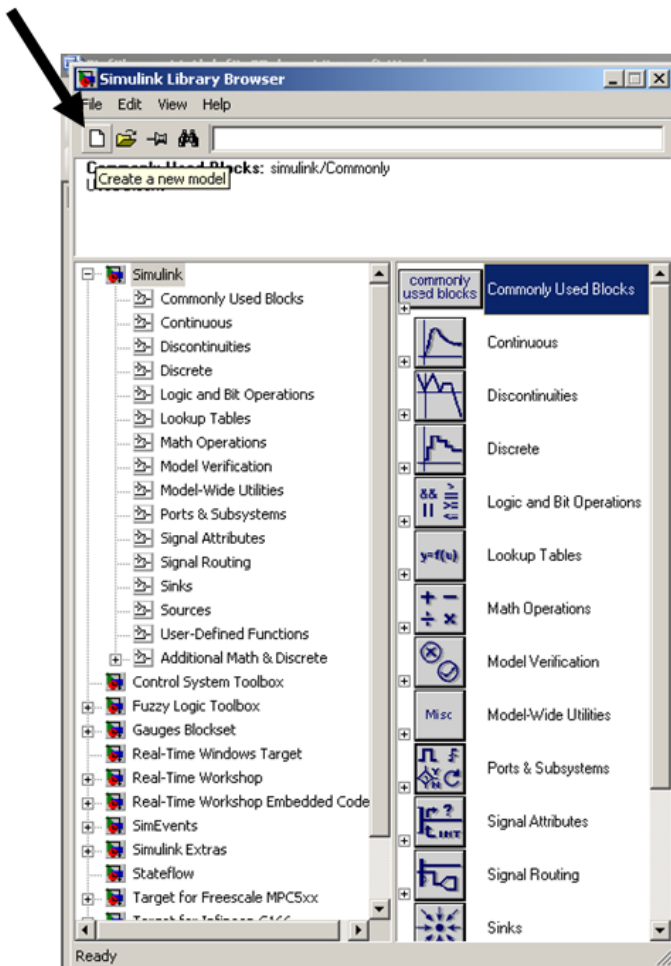
**Achtung:** Der Algorithmus der Funktion „place()“ erlaubt maximal so viele gleiche Pole wie Ausgangssignale vorhanden sind.

### 3 Das Programmpaket Simulink

Matlab  starten, dann Eingabe von `simulink` in der Befehlszeile oder auf das Simulink-Icon  klicken:

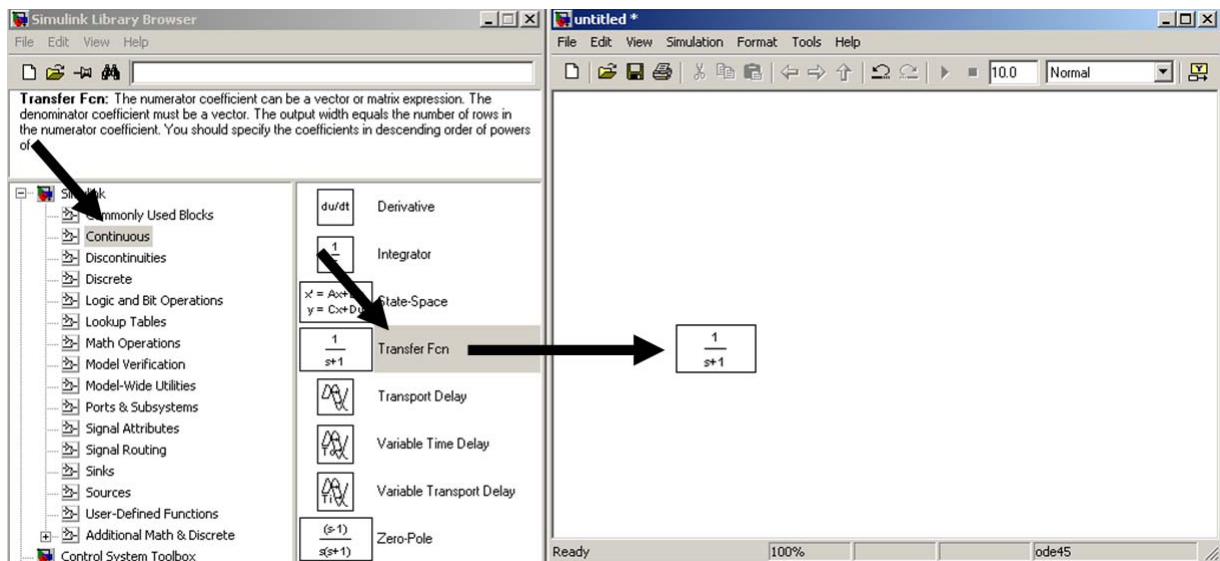


Der Library-Manager wird gestartet. Dort ein „neues Modell“ anlegen:

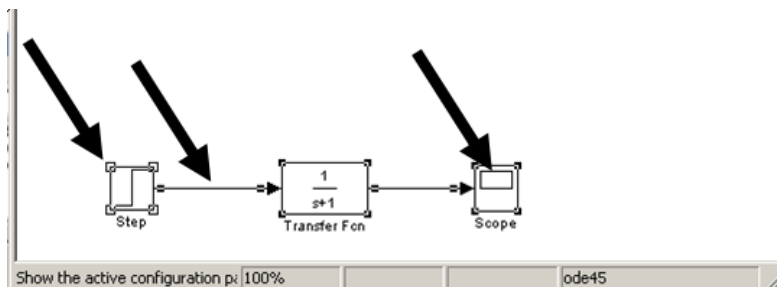


### 3.1 Beispiel

z.B. „Continuous“ anklicken und dann mit „Drag and Drop“ eine „Transfer Function“ in das Arbeitsblatt ziehen.



Ebenso einen „Step“ (Einheitssprung) aus „Sources“ und ein „Scope“ (Oszilloskop) aus „Sinks“ in das Arbeitsblatt ziehen. Dann die Blöcke verbinden: Mit der Maus auf den Ausgang des ersten Blocks links-klicken, gedrückt halten, Verbindung zum Eingang des zweiten Blocks ziehen und dann Taste loslassen.

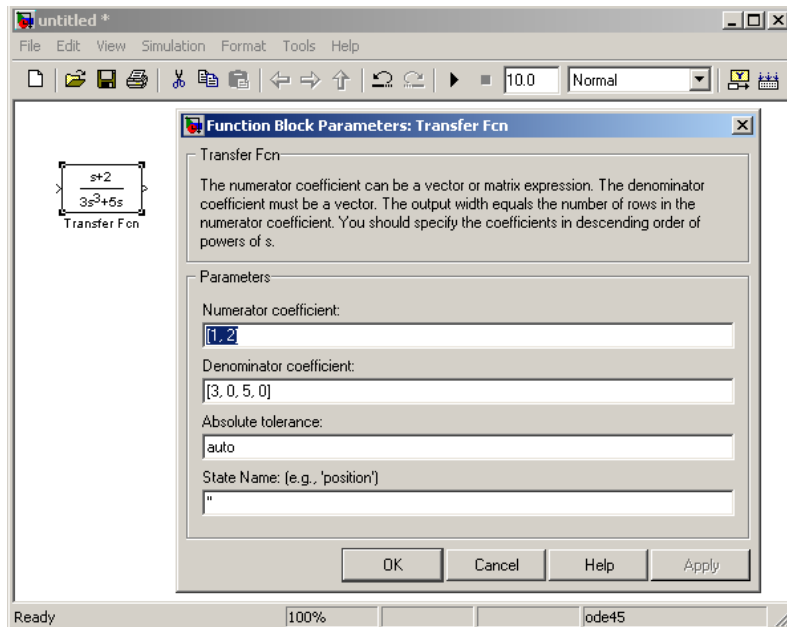


### 3.2 Übersicht Gruppen

- Continuous: Übertragungsfunktion, Totzeit, ...
- Discontinuities: Sättigung, 2-Punkt-Regler
- Discrete: zeitdiskrete Übertragungsfunktion
- Lookup Tables: (nichtlineare) Kennlinien, Kennfelder
- Math Operations: Additionsstellen, Verstärkungsfaktoren, Schieberegler, ...
- Signal Routing: Signale bündeln („Mux“ → Mehrere Signale zu einem Bus-Signal zusammenfassen; nötig für „Mehrkanałoszilloskop“)
- Sinks: Senken (meist Anzeigen) wie Display, Scope (Oszilloskop), ...
- Sources: Quellen wie Step (Sprung), Constant, Sine Wave, ...
- weiter unten: Simulink Extras → Additional Linear: PID-Regler  
Hinweis: Ab Version 2010(?) findet sich der PID-Regler unter „Continuous“.

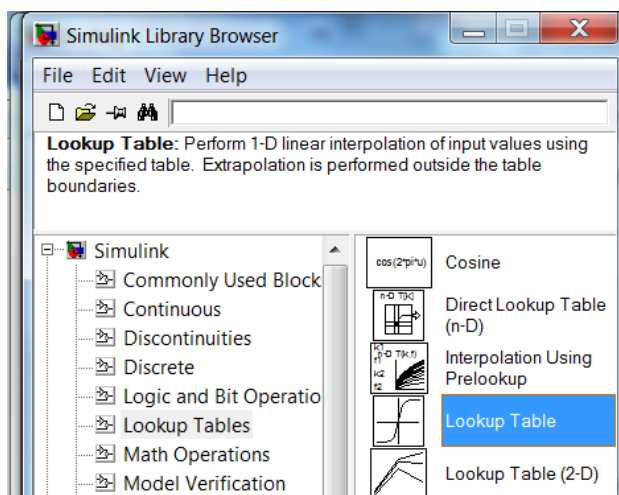
### 3.3 Übertragungsfunktion

Die Zähler- und Nennerpolynome der Übertragungsfunktion/Transfer Function können nach Doppelklick auf den Block in Vektorform eingegeben werden. Dabei entspricht z.B.  $[3, 0, 5, 0]$  dem Polynom  $3 \cdot s^3 + 0 \cdot s^2 + 5 \cdot s + 0 \cdot s^0 = 3 \cdot s^3 + 5 \cdot s$

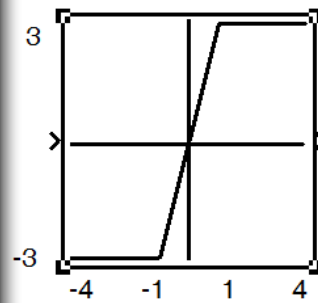
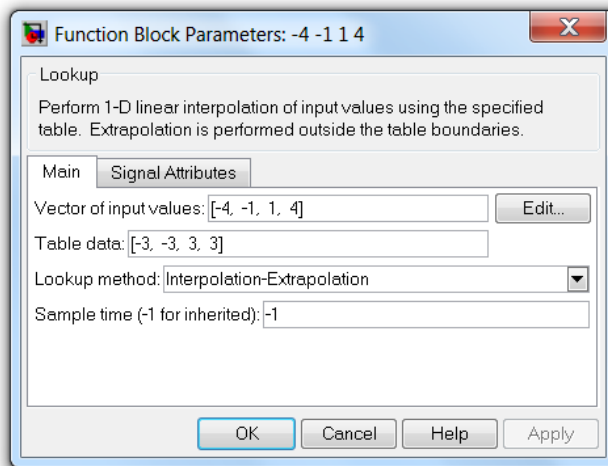


### 3.4 Nicht-lineare Kennlinien

Eine nicht-lineare Kennlinie (z.B. zur Nachbildung von Reibeffekten) findet man unter „Lookup Tables“:



Eine Kennlinie mit den (x, y) Stützpunkten: (-4, -3), (-1, -3); (1, 3) und (4, 3) wird (nach Doppelklick auf das Symbol) folgendermaßen eingegeben:



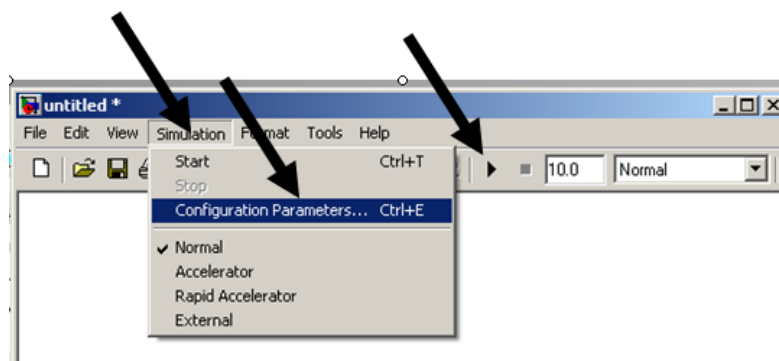
Nach Bestätigung mit „ok“ skizziert Simulink die eingegebene Kurve auf dem „Lookup Table-Block“ (ohne Angabe der Achsenskalierung). Die x-Werte („input values“) müssen monoton ansteigend angeordnet werden. Um Probleme bei der Simulation zu vermeiden, ist es sinnvoll, sie *streng* monoton ansteigend zu wählen (also keine „doppelten“ x-Werte bzw. keine senkrechten Kennlinienteile).

### 3.5 Ändern der Orientierung eines Blocks

Um z.B. bei Rückführzweigen die Signalrichtung des Blocks zu tauschen, verwendet man den Befehl „Flip Block“ (Block rechtsklicken und im Kontextmenü „Format“ auswählen).

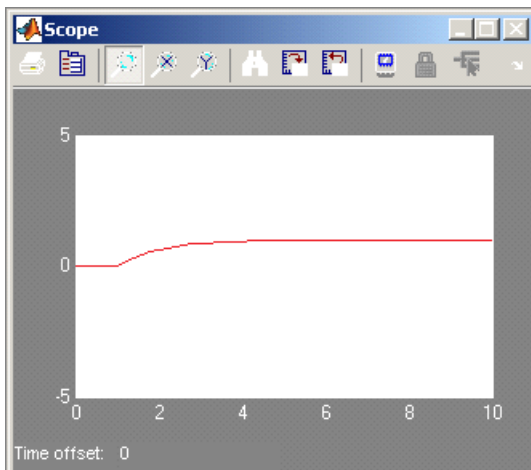
### 3.6 Start Simulation

Bei Bedarf im Menü „Simulation“ und dann „Configuration Parameters“ anwählen und z.B. die Endzeit (Standard: 10 s) verändern. Dann kann die Simulation durch Anklicken des „Play“-Knopfs ▶ gestartet werden.

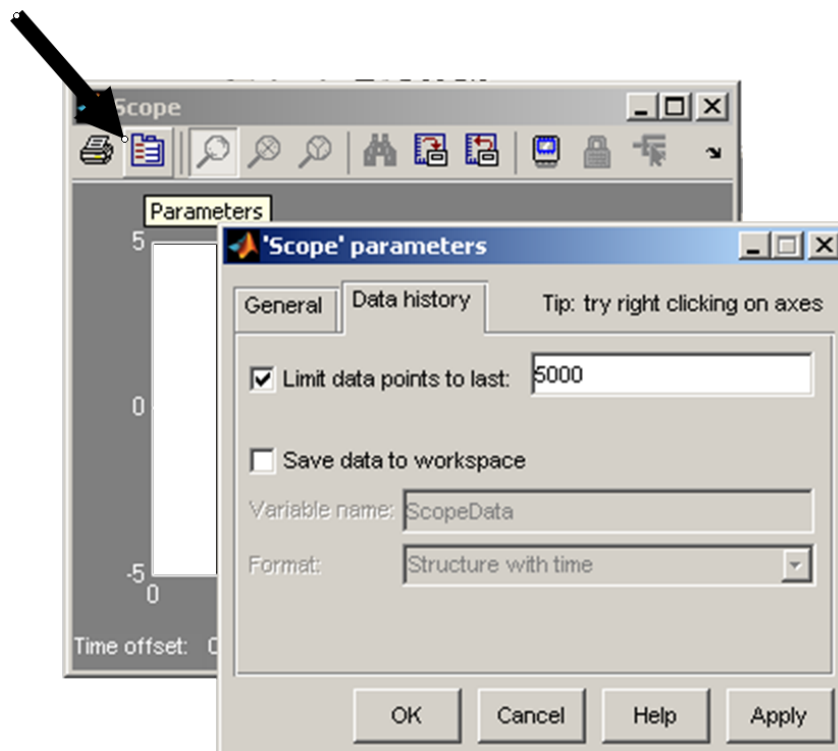


### 3.7 Signalverlauf darstellen (Scope)

Nach Ablauf der Simulation können die Signalverläufe durch Anklicken des Scopes angesehen werden.



Falls die Kurve nicht über dem ganzen gewählten Zeitbereich dargestellt wird, klickt man auf das Parameter-Icon und deaktiviert unter „Data history“ das Häkchen vor „Limit data points to last ...“



### 3.8 Speichern

Modelle können in Windows-üblicher Weise gespeichert und geladen werden. Dabei macht es Sinn, das Current Directory entsprechend einzustellen. (S. Kap. 1.12)



## 4 Preis und Konkurrenzprodukte

Eine Studentenversion von Matlab/Simulink (*MATLAB, Simulink, Symbolic Math Toolbox, Control System Toolbox, Signal Processing, Toolbox Signal Processing Blockset, Statistics Toolbox, Optimization Toolbox, Image Processing Toolbox*) ist für ca. 90 € verfügbar. <http://www.mathworks.com>

Unter <http://www.scilab.org> kann als Alternative die am INRIA/Frankreich entwickelte und frei verfügbare Software Scilab/Xcos (entspricht funktional Matlab/Simulink) heruntergeladen werden.

Unter <http://octave.sourceforge.net/> findet man eine frei verfügbare, fast identische Alternative zu Matlab

## 5 Literatur

A. Angermann et. al.: Matlab – Simulink – Stateflow. Oldenbourg Verlag, München, 2002, ISBN 3-486-25979-2