Module der Automatisierungstechnik

Prof. Dr.-Ing. S. Zacher

info@szacher.de

MATLAB-Grundlagen

Inhalt:

Quellen:

1 Zacher, S., Reuter, M.: *Regelungstechnik für Ingenieure*, Springer-Vieweg Verlag 15. Auflage, 2017, Seiten 417-438

2 *Einführung in MATLAB*. Johannes Gutenberg–Universität Mainz
https://www.uni-frankfurt.de/68661486/einfuehrung_in_matlab.pdf

Was ist MATLAB?

Als Simulationswerkzeug für die Lehrveranstaltung wurde MATLAB (Vertreiber MathWorks GmbH Deutschland) gewählt.

Dies von der Industrie und Forschung anerkannte Programm wurde 1970 an den Universitäten von New Mexico und Stanford entwickelt.

Die Software besteht aus einem Basismodul und etlichen Toolboxen für regelungstechnische Anwendungen, wie Control System, Optimization, Signal Processing, Fuzzy Logic, Neural Network u.a. MATLAB verfügt über eine interaktive Benutzeroberfläche und einen Interpreter, so dass die textuellen Befehle direkt ausgeführt und die Quellcode-Dateien abgearbeitet werden können.

Aus MATLAB können andere C-Programme aufgerufen werden. Das Regelkreisverhalten kann offline und online mit Programm-Tools wie *Matlab*, *Simulink* und *Stateflow* analysiert, eingestellt und visualisiert werden.

Nachfolgend wird zuerst auf die Grundbefehle des Basismoduls und die Menüs des MATLAB/Simulink-Programms sowie auf die Befehle der Control System Toolbox eingegangen. Danach werden die grafische Programmierung mit Simulink, die Datenübertragung und die Kommunikation behandelt. Anschließend werde die Grundlagen der Systemaufbau und die Visualisierung an Beispielen erklärt.

Wie die Abkürzung MATLAB (*Mat*rix *Lab*oratory) besagt, ist das Basismodul für die Operationen mit (*m*, *n*) - Matrizen wie Multiplikation oder Eigenwertberechnung geeignet. Da die skalaren Matrizen der Dimension (1, 1) sind, umfasst das Basismodul alle elementaren mathematischen und logischen Funktionen. Nach dem Aufruf des Programms öffnet sich das Fenster des Basismoduls (*Workspace)* und wartet auf eine Eingabe mit einem Prompt ». In diesem MATLAB-Command Fenster wird der Programmtext eingetragen oder die Funktionen aufgerufen. Alle vorher ausgeführten Anweisungen werden in einer Liste gespeichert und können von der *Command-History* in das MATLAB-Command Fenster kopiert werden.

Die Blockset-Erweiterung von MATLAB ist die Toolbox MATLAB/Simulink, die über eine graphische Oberfläche zur Eingabe von Wirkungsplänen und zur Ausgabe von Simulationsergebnissen verfügt. Die Ergebnisse können durch einen Oszilloskop-Block in das Basismodul übertragen und dort weiter bearbeitet werden.

Die Toolboxen oder die Hilfe dazu kann man durch die Eingabe im Workspace aufrufen, z. B.

» simulink oder » help fuzzy

2 Grundlagen MATLAB

2.1 Variablen und Datentyp

Die Variablen sind Zeilenvektoren (1, *n*), Spaltenvektoren (*m*, 1) und Matrizen (*m*, *n*). Matrizen werden durch eckige Klammern umrahmt dargestellt, die Spalten werden dabei durch ein Komma, ein Leerraum oder einen Zeilenvorschub voneinander getrennt, z. B. für eine Matrix mit $m = 2$ und $n = 3$

$$G = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{vmatrix}$$

gilt:

```
» G = [ a11, a12, a13;  a21, a22, a23];
```

oder

```
» G = [   a11   a12   a13
          a21   a22   a23 ] ;
```

Wie bei den Feldern üblich, kann auf die Elemente einer Matrix durch Indizes zugegriffen werden. Durch Eingabe *G*(1, 2) wird z. B. das Element *a12* aufgerufen.
Die Variablen dürfen aus 31 Zeichen bestehen, das erste muss eine Buchstabe sein. Es wird zwischen Groß- und Kleinbuchstaben unterschieden, was mit

```
» casesen off   oder   » casesen on
```

unterdrückt oder aktiviert werden kann. Nach jedem Befehl werden die Ergebnisse ausgegeben, es sei denn, sie sind mit einem Semikolon abgeschlossen, z. B.

```
» y = 5 * sin (4 * pi * t);
```

Die Konstante *pi* ist vordefiniert. Auch imaginäre Zahlen sind durch Variablen *i* oder *j* vordefiniert, was die Operationen mit komplexen Zahlen durchführen lässt, z. B. die Summe von zwei komplexen Zahlen $s_1 = a_1 + jb_1$ und $s_2 = a_2 + jb_2$:

```
» s1 = a1 + i * b1;   s2 = a2 + i * b2;   s3  =  s1 + s2;
```

Von den vordefinierten Variablen soll noch *eps* erwähnt werden. Sie besitzt einen Wert von 2.2204e - 016, ist damit sehr klein und wird benutzt, um die nicht zugelassenen Operationen wie *Dividing by zero* zu vermeiden.

Standardmäßig sind alle Variablen vom Datentyp *Double Real* mit 64 Bit (Fließkommazahlen mit doppelter Genauigkeit). Ohne Formatierung werden die Zahlen normalerweise mit 4 Nachkommastellen ausgegeben, es sei denn, dass der auszugebende Wert zu klein ist. In diesem Fall wird automatisch auf die Ausgabe mit Exponent umgeschaltet. Die Ausgabe von Zahlen kann man auch mit dem Befehl *format* ansteuern:

```
» format long e; omega
```

wird die Variable *omega* in Exponentenform und mit dem Befehl

```
» format long; phase
```

die Variable *phase* mit 14 Nachkommastellen ausgegeben.

2.2 Befehle und Funktionen

Die mathematischen Ausdrücke werden in Ausgabevariablen gespeichert, z. B.:

```
» a = 2.4;
» b = a + 1.2
```

oder mit der vordefinierten Variable *ans*, wie *answer*, ausgegeben, z. B.

```
» a
ans =
      2.4
```

In einer Zeile können mehrere Befehle eingegeben werden. Der Übertrag eines Ausdrucks in die nächste Zeile erfolgt durch Eingabe von drei oder mehreren Punkten:

```
» Re = –D * cos (alfa);  Im = omega * (1–…
        D) * 0.4;
```

```
>> pi
ans =
        3.1416
>> pi = 3;
>> pi
pi =
         3
```

Für nachfolgende Beispiele mit Matrizen-Operationen sollen zuerst $m = 2$ und $n = 3$, sowie die Matrizen G =[1, 2, 3; 4, 5, 6] und Q = [1, 2; 3, 4] eingegeben werden.

Matrixfunktion	MATLAB-Befehl	Ausgabe
Dimension	» size (G)	» ans = 2 3
Rang	» rank (G)	» ans = 2
Diagonale	» diag (G)	» ans = 1 5
Determinante	» det (Q)	» ans = –2
Inverse Matrix	» inv (Q)	–2.0000 1.0000 1.5000 –0.5000
Transponierte Matrix	» G'	» ans = 1 4 2 5 3 6

In einer Zeile können mehrere Befehle eingegeben werden. Der Übertrag eines Ausdrucks in die nächste Zeile erfolgt durch Eingabe von drei oder mehreren Punkten:

```
» Re = –D * cos (alfa); Im = omega * (1–…
      D) * 0.4;
```

Eine Übersicht der im Programm vorhandenen Variablen wird mit dem folgenden Befehl in Form einer Liste erstellt:

```
» who
```

Einheitsmatrix der Dimension (*m, m)* bzw. size(Q)	» eye (Q)	» ans = 1 0 0 1
Spezielle (*m*, *n*)-Matrix, die nur Einsen enthält	» ones (m, n)	1 1 1 1 1 1
Spezielle (*m*, *n*)-Matrix, die nur Nullen enthält	» zeros (m, n)	0 0 0 0 0 0
Zufallsmatrix (*m*, *n*)	» rand (m, n)	0.8310 0.0535 0.6711 0.0346 0.5297 0.0077

Wie es bei höheren Programmiersprachen üblich ist, bietet MATLAB die Steuerkonstrukte wie bedingte Anweisungen (*if... else if...else*), Auswahlanweisungen (*switch...case*), bedingte Schleifen (*while...end*) und Zählschleifen (*for...end*) an, z. B.:

```
» for i = 1 : 60
    xk = 2 + c1 * (z1 ^ k) + c2 * (z2 ^ k)
    bar (k, xk, 'w')
    hold on
  end
```

Neben arithmetischen Operationen gibt es Vergleichsoperationen wie < (d. h. *kleiner als*) und logische Operationen &, ~, | (UND, ODER, NOT). Mit diesen Befehlen werden z. B. die Elemente von Matrizen *G1* und *G2* wie folgt gebildet:

```
» A = [ 2, 5;  0, 4 ];          % Eingabe: Matrix A
» B = [ 0, 3;  0, 1 ];          % Eingabe: Matrix B
» G1 = A & B                    % Ausgabe: G1 = [ 0, 1;  0, 1 ]
» G2 = A | B                    % Ausgabe: G2 = [ 1  1;  0  1 ]
```

Die Zeilen mit % sind Kommentarzeilen.

2.3 Erstellen von Programmen

Mit MATLAB kann man einzelne Befehle im interaktiven Modus benutzen oder eine Befehlsfolge wie ein Programm (*Matlab-Skript*) zusammenfassen. Das Programm kann mit dem MATLAB- oder einem beliebig anderen Texteditor geschrieben und in einer *m-Datei*, *mat*-Datei oder *ASCII*-Datei abgespeichert werden.

Die Datei mit der Erweiterung **.m* kann mit dem Menü-Befehl *File/New* erstellt und abgespeichert werden, z. B. *aufgabe.m*. Diese Datei wird dann mit dem DOS-Befehl

```
» aufgabe                                    % Aufruf von m-Datei
```

wieder geladen oder mit Menü-Anweisung *File/Open* geöffnet. Einige nützliche Befehle für die Dateienverwaltung sind unten zusammengefasst.

MATLAB-Befehl	Wirkung des Befehls
» fprint (id, 'info', X)	Ausgabe der Variable *X* aus der ASCII-Datei mit dem Identifikator *id* in eine Textdatei
» id = fopen (aufgabe.dat,'w')	Datei *aufgabe.dat* öffnen
» fclose (id)	ASCII-Datei mit dem Identifikator *id* schließen

Programm-Beispiele

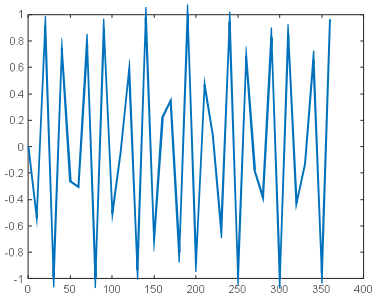
```
clear all; hold off
t=0;
ek=0;
yk=0;
plot(t,yk, 'b*')
hold on
KpR=2;  Tn=20; TA=0.01;
Kid=KpR*TA/Tn;
N=50;
w=ones(N,1);
x=zeros(N,1);
y_stell=zeros(N,1);
for k=1:0.1:N
    ek=1;
    yk=yk+Kid*ek;
    t=t+0.01;
     plot(t,yk,'b*')
end
grid
```

While-Schleife

Rotierende Sinus

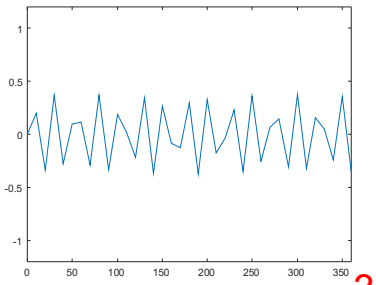
```
close all;
clear all;

x=0:10:360;
y=sin(x);
omega=0;
while 1
  omega=omega+1;
  y_plot=y*sin(omega);
  plot(x,y_plot);

  axis([0,360,-1.2,1.2]);
  pause(0.05);
end
```

plot(x,y)



axis ([0, 360,-1.2, 1.29)]

1.2



-1.2

0

360

Programm mit ctrl+C beenden!

While-Schleife

Sinus-Kaleidoskope

```
close all;
clear all;
hold off
x=0:10:360;
y=sin(x);
omega=0;
while 1
  omega=omega+1;
  y_plot=y*sin(omega);
  plot(x,y_plot);
  hold on
  axis([0,360,-1.2,1.2]);
  pause(0.05);
end
```

3 Grafik mit MATLAB

Ein leeres Grafik-Fenster wird in MATLAB *figure* genannt. Ein Grafik-Fenster kann in mehrere Unterfenster, *subplot*, unterteilt werden. Die Parameter von *figure* sind *Nummer*, die Parameter von *subplot* sind die Koordinaten *zeile, spalte*, *zähler*.

3.1 Grundbefehle

Befehl	Wirkung des Befehls
» figure [(h)];	Ein neues Fenster unter der laufenden Nummer *h* (*handle*) öffnen bzw. ein vorhandenes Fenster Nr. *h* aufrufen
» plot (y);	Die Grafik der Variable *y* erstellen
» subplot (212);	Das Ausgabefenster wird in zwei Teilfenster unterteilt. Position des aufgerufenen Fensters ist: 2. Zeile, 1. Spalte, Zähler = 2
» gcf	Aktuelle Nummer eines Grafik-Fensters (*get handle to current figure*) anzeigen
» legend (s, p)	Eine Box für Text-Kommentar mittels *s* (String) auf der *p* Position erzeugen. Es gilt für die Position: 1 oder 2 - rechte oder linke obere Ecke; 3 oder 4 - untere Ecke usw.
» box on	Die Rahmen eines Diagramms erstellen
» box off	Die Rahmen eines Diagramms löschen
» axis [(–5 2 –4 4)];	Einstellen von Bereichen der Koordinatenachsen (x_{min}, x_{max}, y_{min}, y_{max}). Hier ist: $-5 < x < 2$ und $-4 < y < 4$

» titel ('Bode-Diagramm');	Überschrift der Grafik
» text (–2, 1, 's1');	Beschriftung in einer Grafik: den Text *s1* unter $x = -2$ und $y = 1$ positionieren
» xlabel ('Re'); » ylabel ('Im');	Beschriftung der Achsen. Hier ist *Re* für die x-Achse und *Im* für die y-Achse
» grid	Gitternetz anzeigen
» hold on	Eine neue Grafik zu einer vorhandenen Grafik hinzufügen (Überlappung)
» hold off	*hold on*-Betrieb abschalten
» clf	Aktuelle Fenster löschen (*cleare current figure*), früher » clg
» delete (figure(2));	Fenster Nr. 2 löschen
» close (3); oder » close all;	Fenster Nr. 3 oder alle Fenster schließen

Die Skalierung kann mit *semilogx* und *semilogy* im logarithmischen Maßstab der *x*- und *y*-Achse sowie die Ausgabe in Polarkoordinaten mit *polar(Winkel, Radien, Optionsparameter)* erfolgen. Mit *loglog* werden die beiden *x*- und *y*-Achsen logarithmisch skaliert. Für die graphische Darstellung zweidimensionaler Daten gibt es folgende Möglichkeiten: Balkendiagramm *bar (x, y)*, Liniendiagramm *stem (x, y, format)*, Treppenkurve *stairs (x, y)*, Histogramm *hist (x, y)*, Fehlerintervall *errorbar (x, y, l, u)*.

3.2 Grundformen des *plot*-Befehls

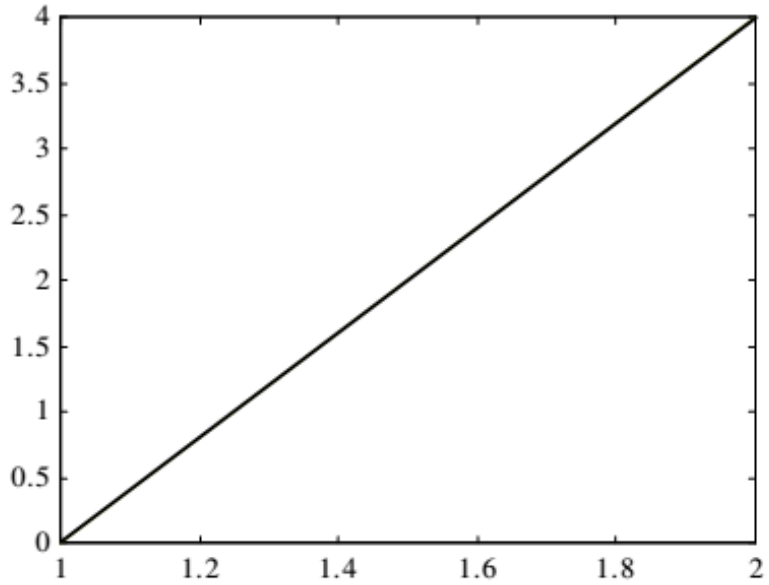
Es gibt mehrere Optionen des *plot*-Befehls, einige davon sind unten aufgeführt.

- Mit dem Befehl *plot(Y)*, wobei *Y* eine (*m*, *n*)-Matrix ist, werden die *n* Spalten als Vektoren betrachtet und die *m* Vektoren hintereinander dargestellt.
- Sind *x* und *y* zwei Vektoren gleicher Art, z. B. zwei Spaltenvektoren (*n*, 1) mit Elementen $x_1, x_2, \ldots x_n$ und $y_1, y_2, \ldots y_n$, so werden mit dem Befehl *plot(x, y)* die Punkte mit Koordinaten (x_k, y_k) durch Linien verbunden.
- In der Grundform *plot(t, y)* wird die Variable *y(t)*, z. B. $y = \sin(2\pi t)$, grafisch dargestellt.
- Mit dem Befehl *plot(t, y1, t, y2)* kann man zwei Signale *y1(t)* und *y2(t)* in einem Bild darstellen, allerdings sollen die Farbe, Art der Linien oder die Markierung der Punkte zusätzlich gewählt werden.
- Der Befehl *plot([y1', y2'], t)* erstellt eine Grafik, die die Ausgabe des Befehls *plot(t, [y1', y2'])* um 90° gedreht darstellt (gilt nur ab Version 5).
- Ist *Z* eine Matrix mit komplexen Elementen, so wird mit dem Befehl *plot(Z)* der Imaginärteil abhängig vom Realteil in der komplexen Ebene abgebildet, z. B.

```
» a1 = 1;   b1 = 2;
» Z = [a1    i * b1];
» plot (Z)
```

3.3 Darstellung eines Vektors

Im Fall $y = [a_{11}]$ wird mit *plot(y , ' *')* ein Punkt * unter $x = 1$ und $y = a_{11}$ positioniert. Im Fall $n = 2$ (zwei Spalten) wird der Vektor $y = [a_{11}, a_{12}]$ wie in **Bild 14.1** gezeigt abgebildet.



Vektor $y = [a_{11}, a_{12}]$ mit zwei Spalten

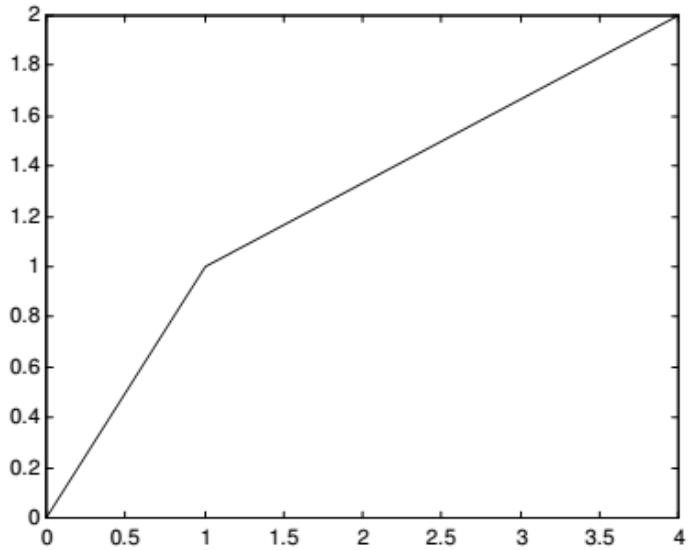
```
» % Befehle
» y = [0  4] ;
» plot (y)
```

3.4 Farbzeichen und Linientypen

Für Variablen *x*, *y* können mit dem *plot* - Befehl die Farbe, der Linientyp und die Markierung der Punkte eingestellt werden.

Farbzeichen		Linientypen	Punkten-Markierung
'b' = blue	'm' = magenta	' - ' = solid	'+'
'c' = cyan	'r' = red	'--' = dashed	'-'
'g' = green	'w' = white	' : ' = dotted	'--'
'k' = black	'y' = yellow	'-.' = dash-dot	'*'

Bild zeigt die Ausgabe eines Programms, das die Punkte mit den Koordinaten ($x1$, $y1$), ($x2$, $y2$) und ($x3$, $y3$) durch Geraden mit Farbe *black* verbindet.

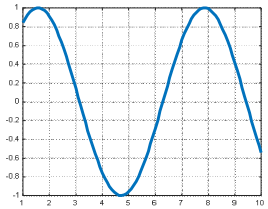



```
» % Befehle
» x = [0  1  4] ;
» y = [0  1  2] ;
» plot (x, y, ' k ')
```

Verbindung von 3 Spalten bzw. Punkten (0, 0), (1,1) und (4,2)

3.5 Kurven

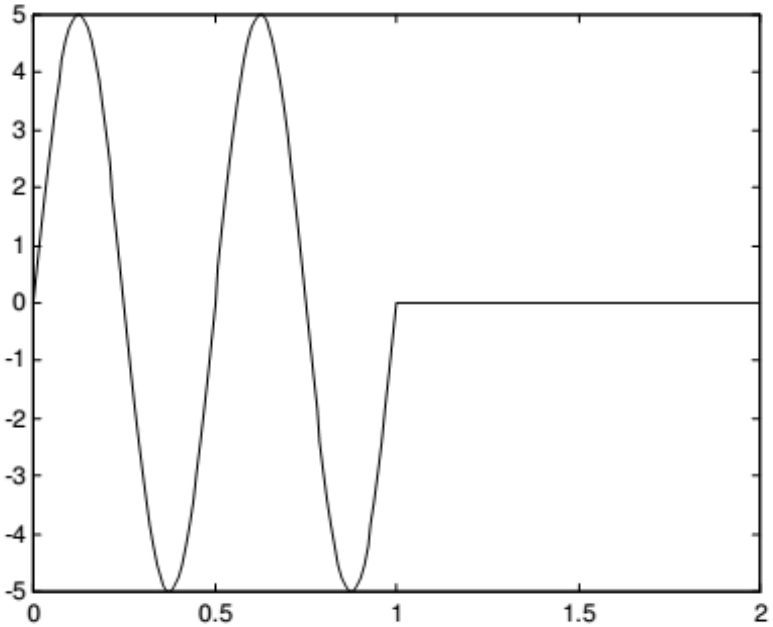
```
x = linspace(1,10);
plot(x,sin(x))
grid
```



Die Eingabe einer Zeitspanne erfolgt durch die Anweisung

```
» t = 0:delta:max,
```

wobei *delta* für die Schrittweite und *max* für die rechte Grenze des Zeitrasters steht. In **Bild** ist die Ausgabe eines Programms gezeigt, das erst im Zeitraster $0 < t < 1$ ein harmonisches Signal *y(t)* erzeugt und dann zu einer horizontalen Linie übergeht.



Verknüpfung von zwei Signalen

```
» % Matlab-Programm
» t = 0:0.01:1;
» y1 = 5*sin(4*pi*t);
» plot (t, y1);
» hold on;
» x = [0  0] ;
» plot (x,  'k');
```

3.6 3D-Kurven

Erweitert man den plot-Befehl zu

```
» plot3 ( x, y, z, 'k * ' );
```

so wird die Grafik dreidimensional mit schwarzen Punkten * ausgegeben. Ein Beispiel der nichtlinearen statischen Kennlinie, die für 50 Punkte berechnet wird,

```
» x=(0:50) / 10;
» plot3 ( ( 1-x ), ( 3*x ), ( 1-0.5*x ).*( 1-0.5*x ),' k- ' )
```

ist in **Bild 14.5** gezeigt.

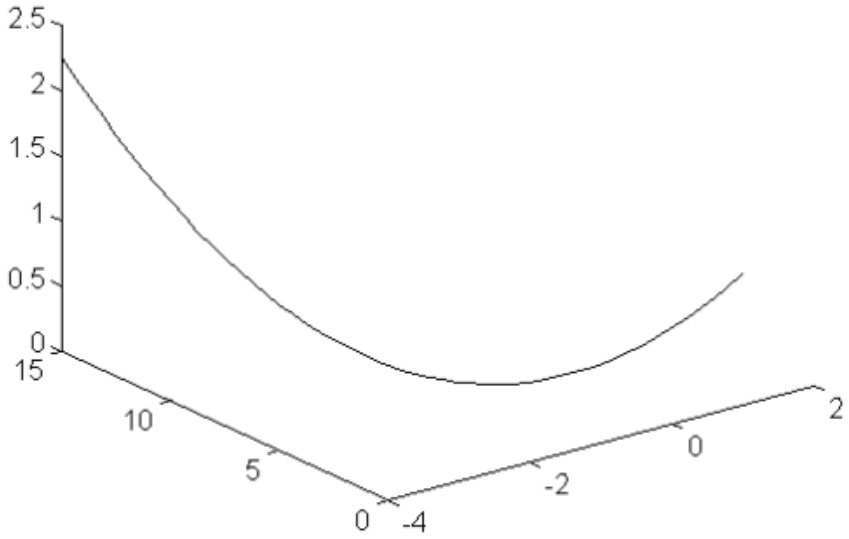


Bild 14.5 3D-Grafik mit dem *plot3*-Befehl

Mit dem Befehl

```
» [ X, Y ] = meshgrid (x, y);
```

wird aus Variablen *x* und *y* eine Matrix berechnet, deren Zeilen und Spalten die Vektoren *x*, *y* sind.

Eine Funktion, z. B.

```
W = (Y – 1).* (X – 1) + Y.* X;
```

kann mit folgenden Grafikfunktionen als Maschen- und Kachelplots dargestellt werden:

» contour (X, Y, W, N);	2D-Darstellung mit *N* Konturlinien
» contour3 (X, Y, W);	3D-Höhenlinienplot (perspektivisch)
» mesh (X, Y, W);	3D-Gitterdarstellung
» surf (X, Y, W);	3D-Flächen (Kachelplot)

4 Beispiele von Applikationen

4.1 Matrix A um eine Zeile / Spalte erweitern / löschen

```
>> A = [ –1.5, 3, 0.5; –2, 2.8, 1; 5, 4, 1.3 ]          % Eingabe: Matrix

>> A = [A; 1, 2, 3]      % Eingabe: Matrix A um eine Zeile erweitern

>> A = [A, [3; 4; 5] ]          % Eingabe: A um eine Spalte erweitern
```

oder man kann eine Spalte mit

```
S4 = [ 3; 4; 5 ]
```

eingeben und dann die Matrix um diese Spalte erweitern:

```
A = [ A, S4]
```

Will man die 4. Spalte wieder herauslöschen, so muss die 4. Spalte mit einer leeren Spalte [ ] belegt werden. Die 4. Spalte wird mit dem Index 4 angesprochen. Da der Index der Zeile in diesem Fall beliebig ist, wird der Index mit dem Platzhalter : gekennzeichnet:

```
A ( : , 4 ) = [ ]
```

Die 1. Zeile wird nach diesem Prinzip mit der Anweisung

```
A ( 1 , : ) = [ ]
```

gelöscht.

4.2 Lösung von linearen Gleichungssystemen

Aufgabe

$$\begin{cases} -x + y + z = 0 \\ x - 3y - 2z = 5 \\ 5x + y + 4z = 3 \end{cases}$$

Lösung: $ax = c$

oder $x = a \wedge (-1) * c$

oder $x = \text{inv}(a) * c$

MATLAB-Skript:

```
>> A = [–1, 1, 1; 1, –3, –2; 5, 1, 4]      % Eingabe
```

```
>> C = [0; 5; 3]                            % Eingabe
```

```
>> inv (A) * C                              % Eingabe
```

4.3 Vektoralgebra

Vektoraddition bzw. Subtraktion:

Gegeben: A = [1, –3, 5] und B = [2, 1, –4];

Eingabe: C = A + B % Ausgabe: C = 3 –2 1

Länge eines Vektors (Betragsbildung):

Cmod = sqrt (C(1)^2 + C(2)^2 + C(3)^2)

bzw. norm (C) %Ausgabe: Cmod = 3.7417

Maximale (minimale) Komponente des Vektors:

[m, index] = max (A) % Ausgabe: ans = 5 index = 3
[m, index] = min (A) % Ausgabe: ans = –3 index = 2

Transponieren (einen Spaltenvektor aus dem Zeilenvektor und umgekehrt):

A = [1; –3; 5]; A' % Ausgabe: ans = 1 –3 5

Multiplikation mit dem Skalar: D = 5 * C

% Ausgabe: D = 15 –10 5

Skalarprodukt: a) Gegeben: A = [1, –3, 5] und B = [2, 1, –4];

F = A * B’; % Ausgabe: F = –21

b) Gegeben: A = [1; –3; 5] und B = [2, 1, –4];

F = A’ * B’; % Ausgabe: F = –21

Positiv semidefinite Matrix Q.

Die Matrix Q ist positiv semidefinit,
wenn alle Eigenwerte der Matrix nicht negativ sind:

$$\lambda_k \geq 0$$

Q=[10 11 12; 1 2 3; 4 5 6];
eig(Q)

$$\lambda_1 = 16.9373$$
$$\lambda_2 = 1.0627$$
$$\lambda_3 = 0.0000$$

Positiv definite Matrix R.

Die Matrix R ist positiv definit,
wenn alle Eigenwerte der Matrix positiv sind:

$$\lambda_k > 0$$

R=[2 0 0; 0 0 2; 0 0 1];
eig(R)

$$\lambda_1 = 1$$
$$\lambda_2 = 2$$
$$\lambda_3 = 2$$

4.4 Lösung von algebraischen Gleichungen

$$p = a_4 x^4 + a_3 x^3 + a_2 x^2 + a_1 x + a_0 = 0$$

$$p = 2x^4 + x^3 + 5x^2 + 9x + 1 = 0$$

$$x_1 = 0{,}4165 + 1{,}8146j$$
$$x_2 = 0{,}4165 - 1{,}8146j$$
$$x_3 = -1{,}2142$$
$$x_4 = -0{,}1188$$

```
a4=2;
a3=1;
a2=5;
a1=9;
a0=1
p = [a4   a3  a2  a1  a0];
x = roots(p)
```

4.5 Ausgabe mehrere Kurven

```
t = 0:0.01:1;
y1=5*sin(4*pi*t);
y2=2*cos(2*pi*t);
y3=4*sin(2*pi*t)+cos(2*pi*t);
```

```
plot(t, y1, t,y2,t,y3);grid;
```

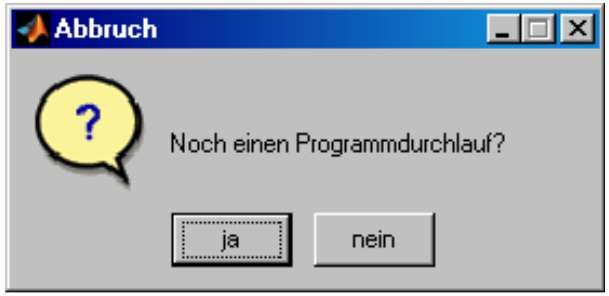
```
subplot(311); plot(t, y1);grid;
subplot(312); plot(t, y2);grid;
subplot(313); plot(t, y3);grid;
```

4.6 Animation: Springender Ball

```
close all; clear all;                                          ← workspace löschen
initpos=50; initvel=0; r_ball=5; gravity=10;                   ← Initialisierung (R=5; g=9,8 m/s^2)
c_bounce=0.9;                                                  ← Dämpfung (c_bounce=0.9)
dt=0.05;                                                       ← dt =Rechenschritt
rectangle('Position', [-r_ball, initpos, r_ball, r_ball],...
'Curvature',[1,1], 'FaceColor','b');                           ← rectangle: Frame mit Kreis
line([-5*r_ball, 5*r_ball], [0,0]);                            ← line: Bodenlinie unten fixieren
axis('equal');                                                 ← equal: Kreis (kein Oval)
pos=initpos-r_ball; vel=initvel;                               ← Umrechnung (Position, Geschwindigkeit)
while 1                                                        ← Beginn Schleife WHILE
      pos=pos+(vel*dt);                                            ← aktuelle Position
      vel=vel-(gravity*dt);                                        ← aktuelle Geschwindigkeit
            if pos<0                                           ← Richtung-Umkehr am Boden (pos<0)
               vel=-vel*c_bounce;
            end
      clf                                                      ← clf: Fensterinhalt löschen
      rectangle('Position', [-r_ball, pos, r_ball, r_ball],... ← aktuelle Frame rectangle
      'Curvature',[1,1],'FaceColor','b');
      line([-5*r_ball, 5*r_ball], [0,0]);                      ← fixierte Bodenlinie line
      axis([-5*r_ball, 5*r_ball, 0, initpos+2*r_ball]);        ← Anpassung von axis-Skalierung
      axis('equal');                                           ← Kreisförmige Ball equal (kein Oval)
      axis('off');                                             ← Skala verstecken off (Hintergrund grau)
      pause(0.02);                                             ← cpu = 0.03; pause(dt-cpu)=0.05-0.03
end                    % Programm mit ctrl+C beenden              Programm  mit ctrl+C beenden
```

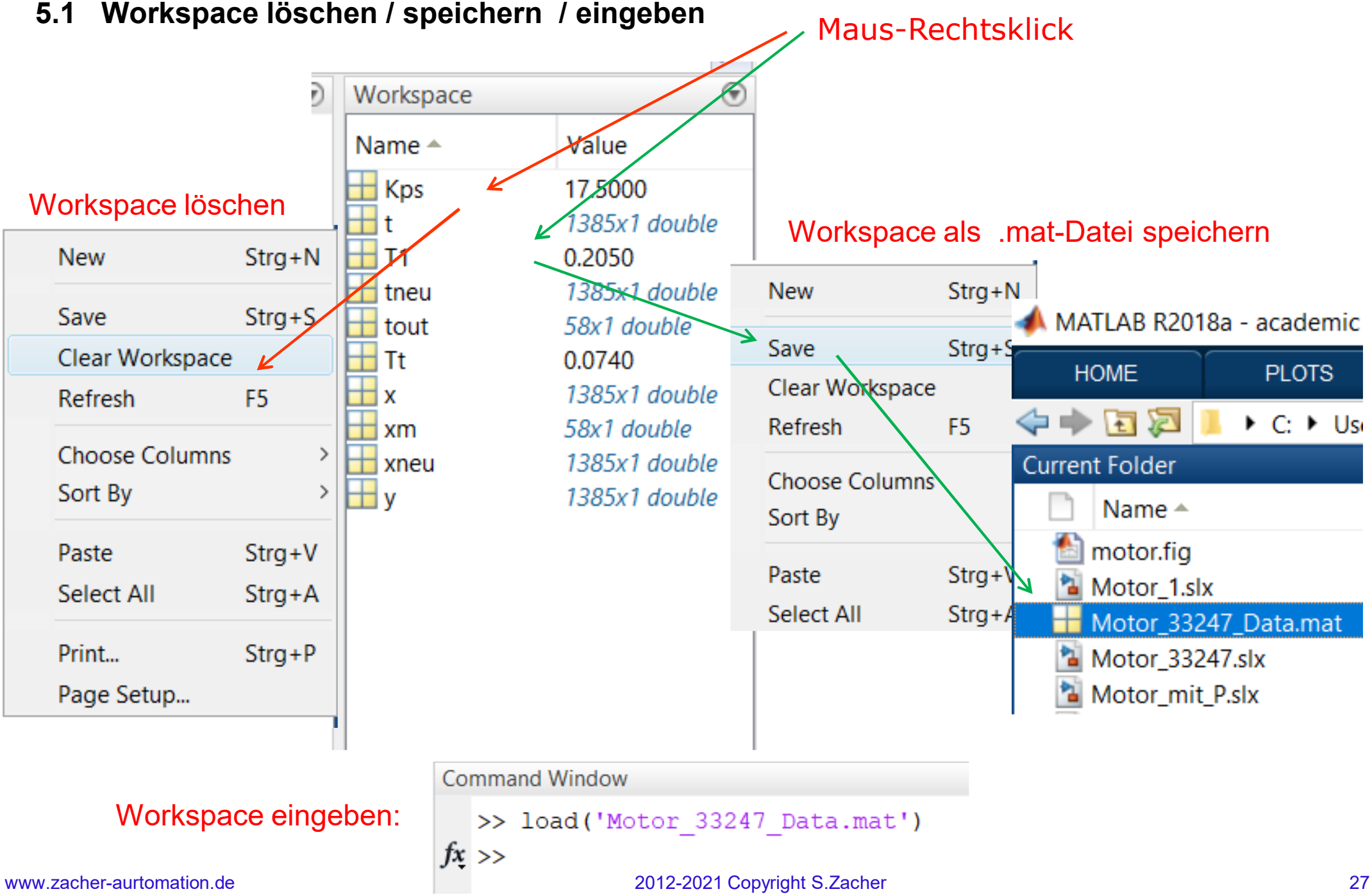
4.7 Programmdurchlauf

```
% Frage nach Programmabbruch durch Dialogbox
durchlauf = 'j';
while durchlauf == 'j'
disp('Programmdurchlauf');
button = questdlg('Noch einen Programmdurchlauf?', 'Abbruch', 'ja','nein', 'ja');
switch button
case 'ja'
durchlauf = 'j';
case 'nein'
durchlauf = 'n';
end;
end
```



5 Datenübergabe

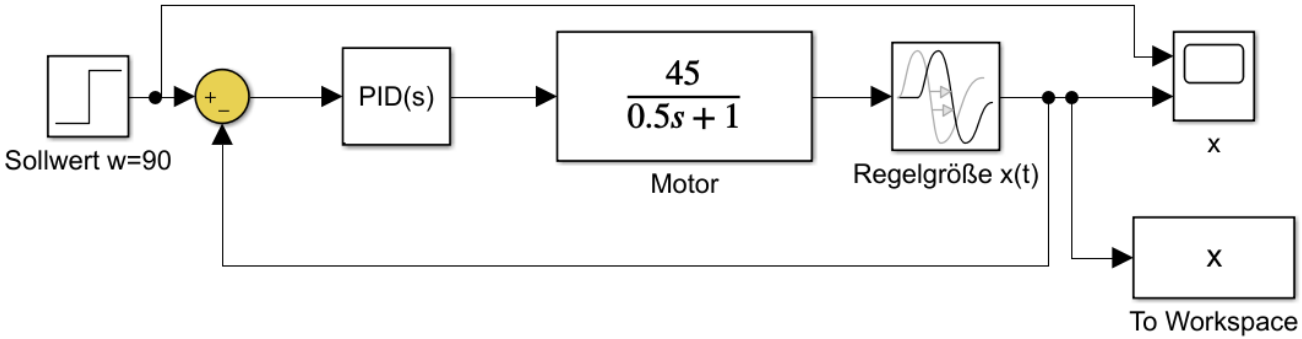
5.1 Workspace löschen / speichern / eingeben



Workspace eingeben:

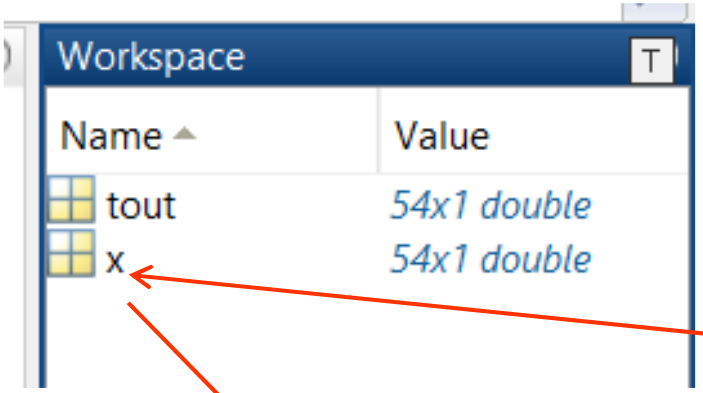
```
Command Window
>> load('Motor_33247_Data.mat')
>>
```

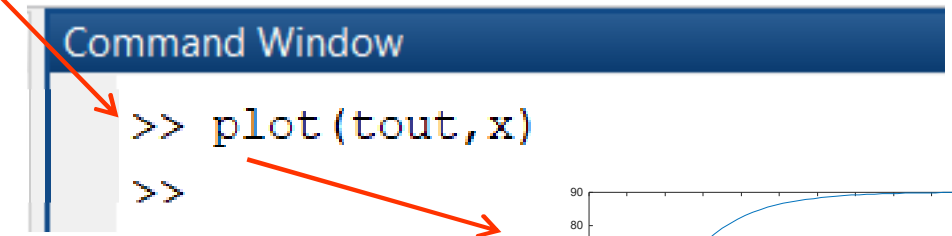
5.2 Datenübergabe von Simulink zu MATLAB : *To Workspace*

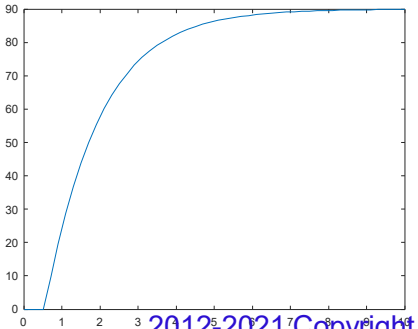


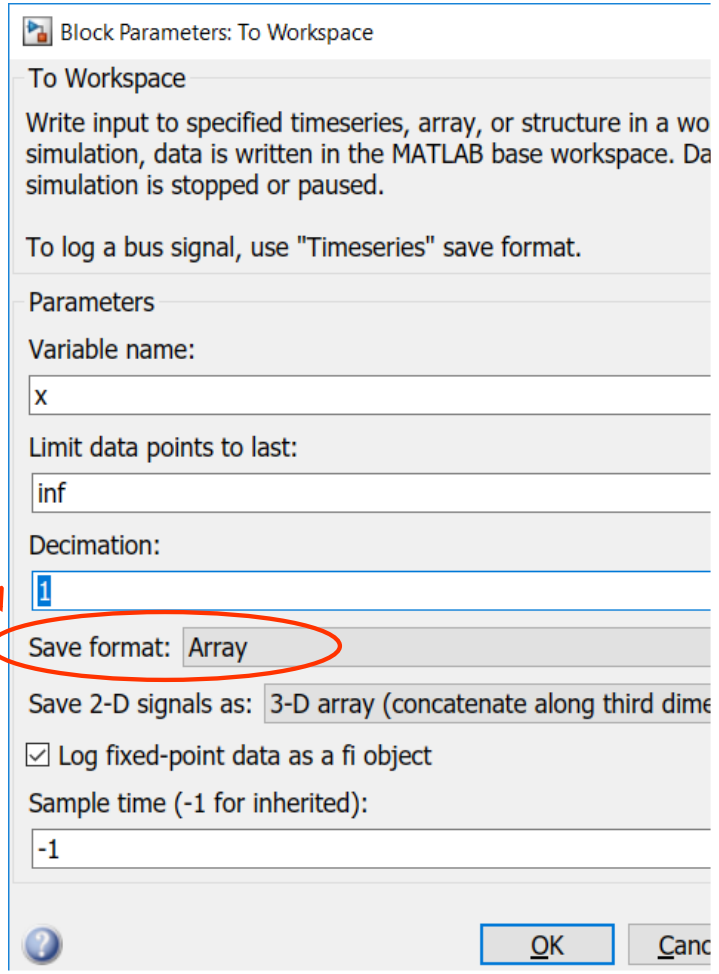
Simulink

MATLAB



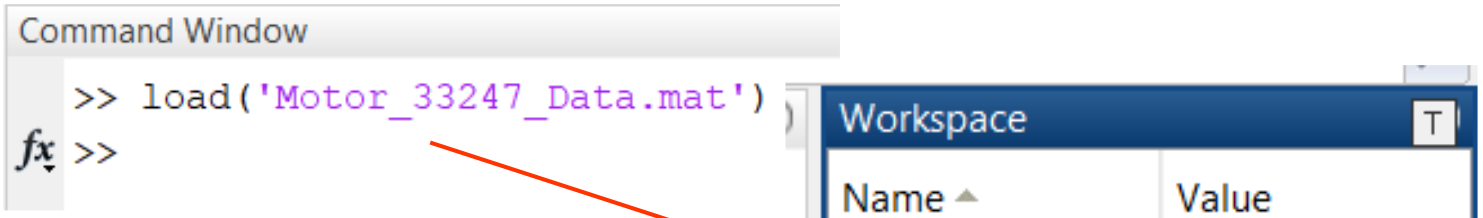


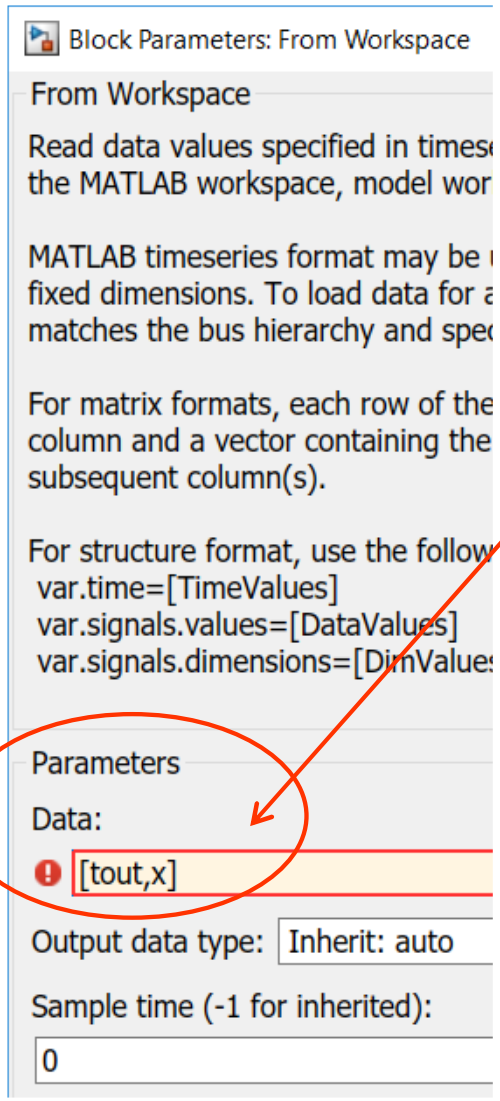




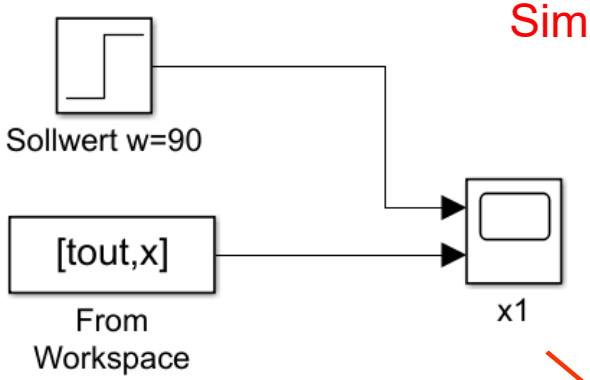
5.3 Datenübergabe zwischen MATLAB und Simulink: *From Workspace*

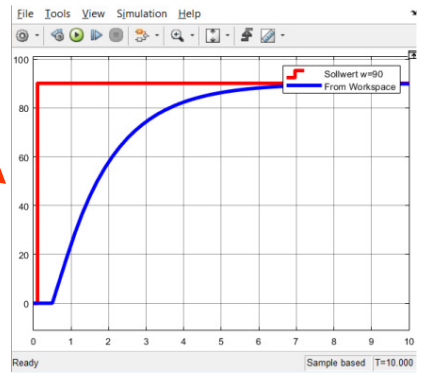
MATLAB





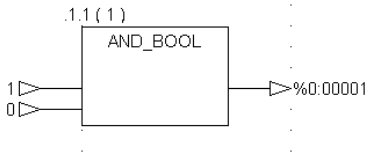
Simulink





5.4 BOOL-Ausgabe

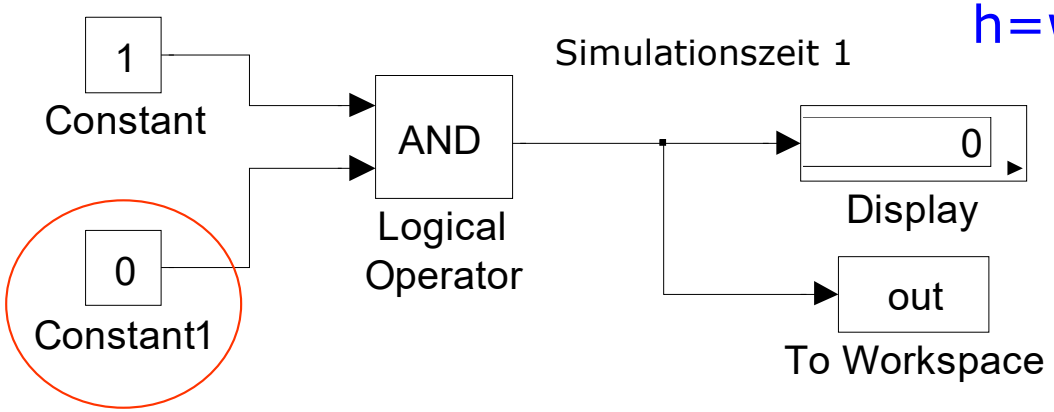
Beispiel: AND-Block

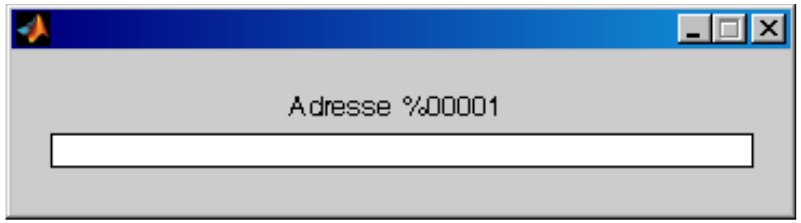


Step 1. Erstellen Sie die Simulink-Datei Aufgabe_1.mdl

Step 2. Starten Sie die Simulation und geben Sie danach den MATLAB-Befehl ein:

```
h=waitbar(int8(out(1)),'Adresse%00001')
```

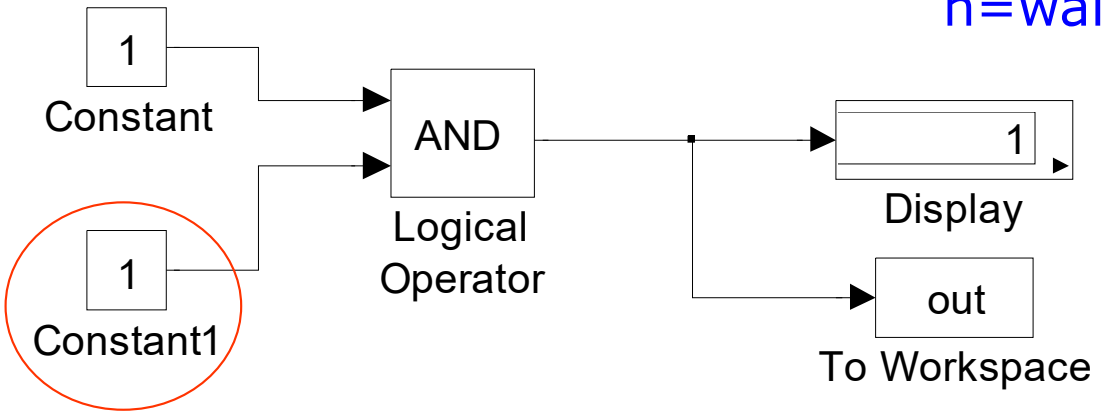


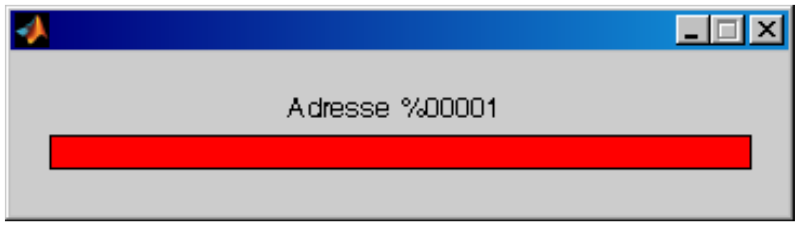


Step 3. Ändern Sie den Eingang *Constant1*

Step 4. Starten Sie die Simulation, dann geben Sie ein:

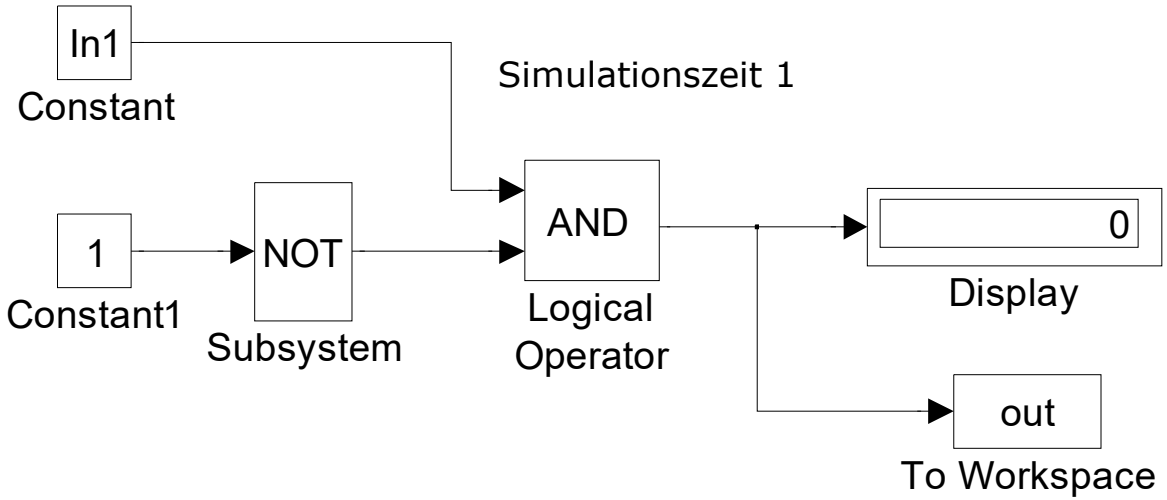
```
h=waitbar(int8(out(1)),'Adresse %00001')
```





5.5 Dialogfenster: Eingabe über MENU-Button

Step 1. Erstellen Sie die Simulink-Datei Aufgabe_2.mdl



Step 2. Öffnen Sie das Dialogfenster mittels MATLAB-Befehls

```
In1 = menu ('EINGABE', 'In1')
```

Step 3. Klicken Sie auf das Button, der Wert In1=1 wird dem Eingang In1 zugewiesen

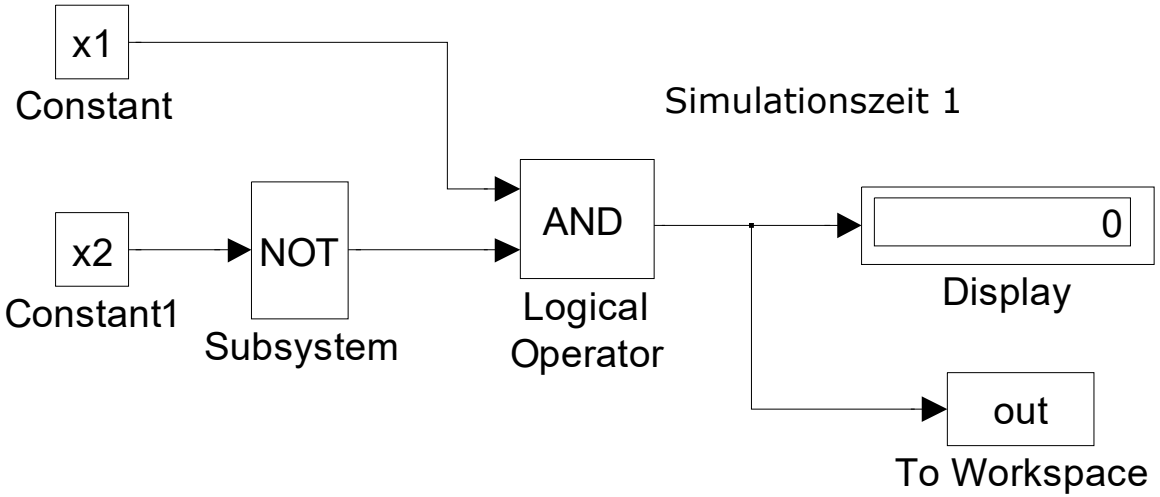
```
>> In1 = menu ('EINGABE', 'In1')

In1 =

     1
```

Step 4. Starten Sie die Simulation der Datei Aufgabe_2.mdl

5.6 Datenübergabe von EXCEL zu MATLAB

MATLAB als Client, Eingabe von Mappe_X.xls

Step 1. Modifizieren Sie die Simulink-Datei Aufgabe_2.mdl



Step 2. Erstellen Sie die EXCEL-Datei Mappe_X.xls mit folgendem Inhalt

Mappe_X.xls

	A	B	C
1	0	0	1
2	1	2	3
3			

Step 3. Verbinden Sie die Datei Aufgabe_2.mdl mit Mappe_X.xls über DDE mit

```
Kanal=ddeinit('Excel','Mappe_X.xls')
```

Es erscheint: Kanal = 4.7836e-299

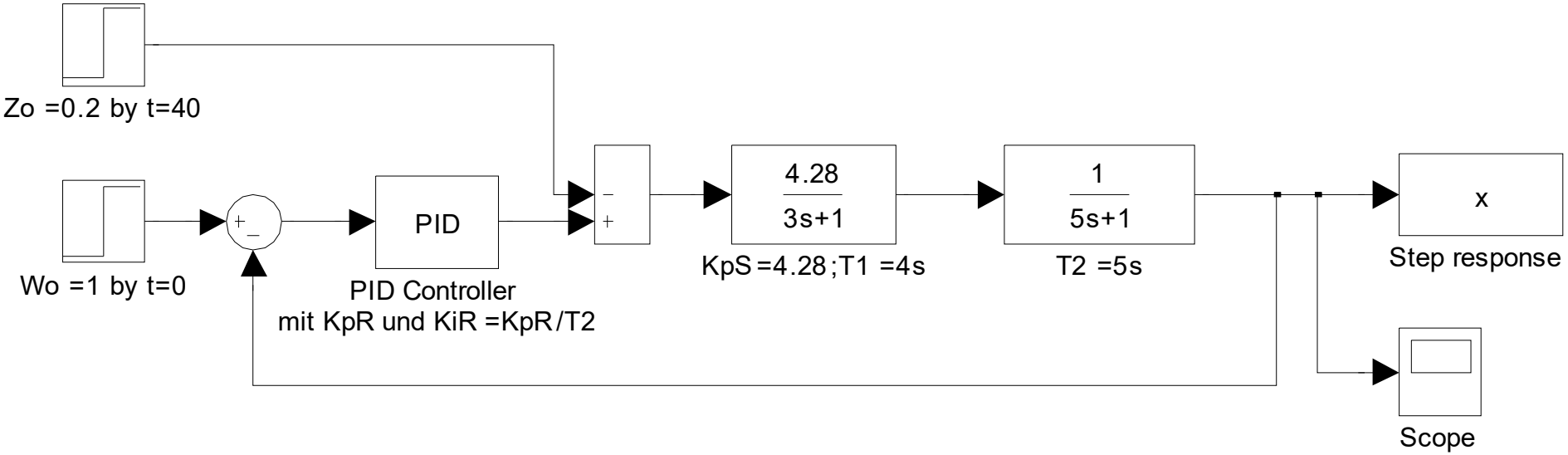
Step 4. Holen Sie die Werte aus der EXCEL-Tabelle, z.B.

```
dat=ddereq(Kanal,'r1c1:r1c3')
```

Step 5. Ordnen Sie die Werte zu Eingängen, z.B. wie unten, und starten Sie die Simulation

```
x1=dat(2);  x2=dat(3);
```

5.7 Datenübergabe von MATLAB zu EXCEL MATLAB als Server, Ausgabe in Mappe_Y.xls



Step 1. Erstellen Sie die EXCEL-Datei Mappe_Y.xls

Step 2. Geben Sie **KpR=0.3** und simulieren Sie den Regelkreis.

Step 3. Verbinden Sie die Datei PI_1.mdl mit Mappe_Y.xls

```
Kanal=ddeinit('Excel','Mappe_Y.xls')
```

Es erscheint: Kanal = 4.7836e-299

Mappe_Y.xls

	A	B	C	D
1	300	30	3	
2	100	10	1	
3				

Step 4. Senden Sie den Wert KpR von MATLAB zu EXCEL

```
rc=ddepoke(Kanal,'r1c4',KpR)
```

Bei Erfolg ist der Rückgabewert **rc=1**, sonst **rc=0**

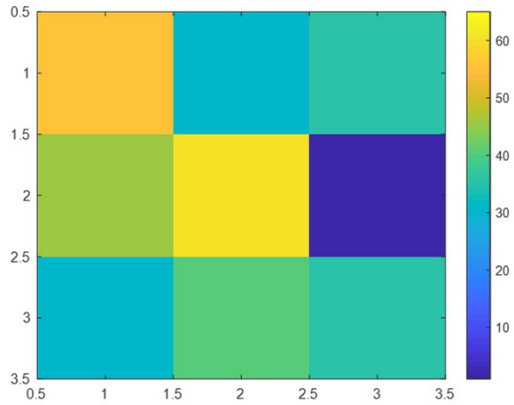
Mappe_Y.xls

	A	B	C	D
1	300	30	3	300
2	100	10	1	
3				

6 Bildbearbeitung

6.1 Ein Bild erstellen

```
M=[55 30 35;45  60  0; 30  40  35];
image(M);
C=colorbar;
```



6.2 Ein Bild einlesen

```
R=imread('planet.jpg');
image(R);
```

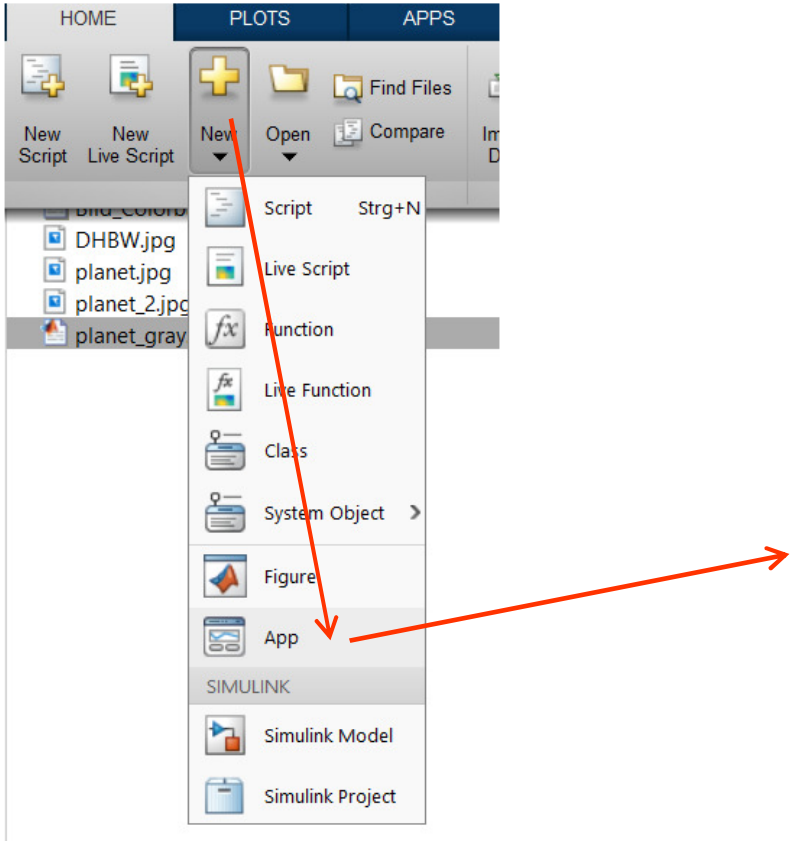
6.3 Ein Bild bearbeiten

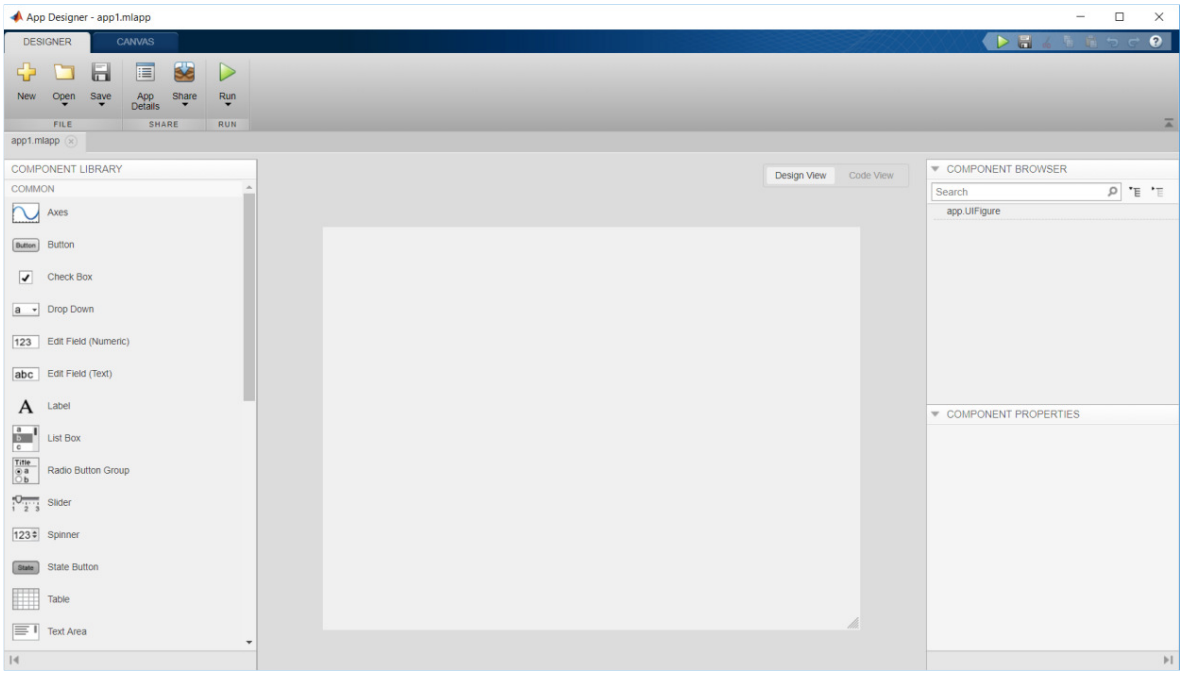
```
I=rgb2gray(R);
h=[1  2  1; 0  0  0; -1 -2 -1];
I2=filter2(h,I);
imshow(I2,'DisplayRange',[ ]);
```

7 App Designer

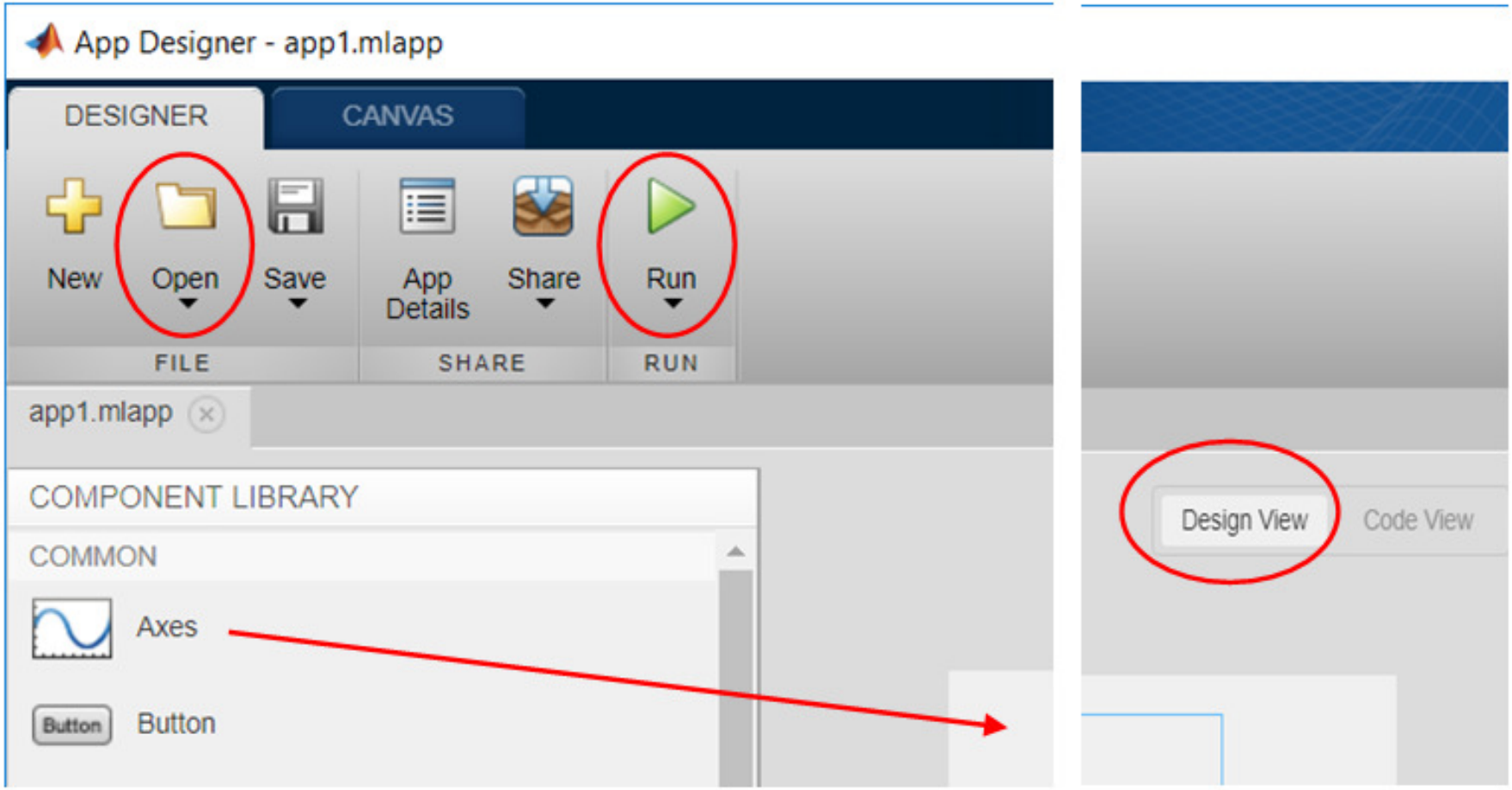
Mit App Designer kann man visuelle Komponenten einfach per Drag and Drop in den Design-Bereich ziehen. Der objektorientierte Code für Layout und Design wird automatisch erzeugt. Die Eingabe und das Editieren des Skripts sind auch einfacher als bei GUIDE.

Erstellen wir als Beispiel eine einfache Applikation. Dafür starten wir App Designer aus dem *MATLAB Command Fenster* mit *HOME* → *New* → *App*.





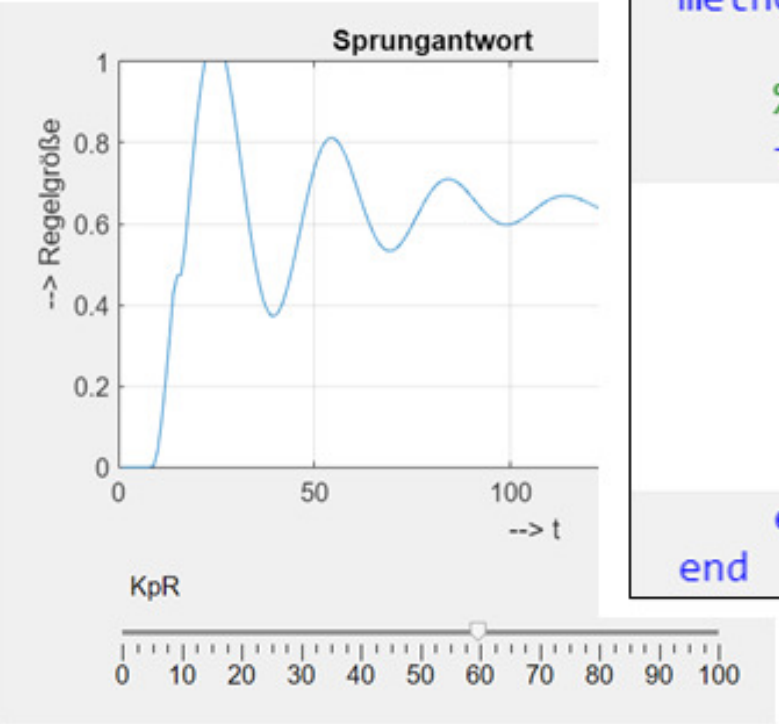
Im eröffneten App Designer Fenster (Bild unten) starten wir mit *Open* → *EXAMPLES* → *Interactive Tutorial* zum Einstieg in die Handhabung der Software.



Nun erstellen wir ein Beispiel *P_kreis.mlapp*. Ziehen wir nacheinander Elemente *Axes* und *Slider* aus dem *COMPONENT LIBRARY* in das *Design View* Fenster. Dann wechseln wir von *Design View* zu *Code View* und fügen eine neue Funktion

```
methods  (Access = private)
```

ein, um die *KpR* des P-Reglers mit dem *Slider* anzusteuern (Bild rechts). Mit dem Klick auf die Taste RUN des App Designers wird das Panel erstellt (Bild links).




```
methods (Access = private)

    % Value changed function: KpRSlider
    function KpRSliderValueChanged(app, event)
        KpR = app.KpRSlider.Value;
        s=tf('s');
        G0=KpR*0.03*exp(-s)/(s^2+2*s+1);
        Gw=G0/(1+G0);
        plot(app.UIAxes,step(Gw))
        app.UIAxes.YLim = [0 1];
    end
end
```

Module der Automatisierungstechnik

Prof. Dr.-Ing. S. Zacher

info@szacher.de

Ende der Präsentation

MATLAB-Grundlagen