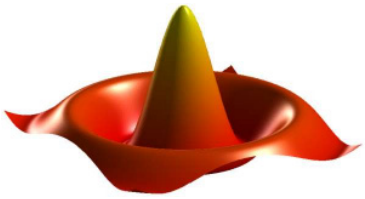
MATLAB – Simulink – Stateflow

Grundlagen, Toolboxen, Beispiele

Angermann/Beuschel/Rau/Wohlfarth



Lösungen der Übungsaufgaben

Stand: 30. März 2005

www.matlabbuch.de

Alle Rechte vorbehalten
Vervielfältigung jeder Art nur
mit Erlaubnis der Autoren

Inhaltsverzeichnis

1 Einführung

Die vorliegende Musterlösung zum Buch *Matlab – Simulink – Stateflow* enthält Lösungsvorschläge zu allen Übungsaufgaben des Buches. Neben dem vollständigen abgedruckten MATLAB-Code bzw. den Grafiken der Simulink Modelle zu den jeweiligen Aufgaben werden notwendige zusätzliche Erklärungen und Kommentare zum gewählten Lösungsweg gegeben. Sämtliche Lösungen sind als MATLAB- bzw. Simulink-Programme auf dieser CD-ROM enthalten, so dass die Lösungsvorschläge direkt gestartet und bearbeitet werden können.

1.1 Verzeichnis- und Dateistruktur der CD-ROM

Zu jedem Kapitel des Buchs, das Aufgaben enthält, existiert ein eigenes Verzeichnis auf der CD-ROM. Es gilt die folgende Zuordnung zwischen den einzelnen Kapiteln und Verzeichnissen:

Kapitel		`Verzeichnisname`
2.	Matlab Grundlagen	`grundlagen`
3.	Eingabe und Ausgabe in Matlab	`inout`
4.	Differentialgleichungen in Matlab	`dgl`
5.	Regelungstechnische Funktionen	`control`
6.	Signalverarbeitung	`signal`
7.	Optimierung	`optim`
8.	Simulink Grundlagen	`simulink_grund`
9.	Lineare und nichtlineare Systeme in Simulink	`simulink_systeme`
10.	Abtastsysteme in Simulink	`simulink_abtast`
11.	Regelkreise in Simulink	`simulink_regelkreis`
12.	Stateflow	`stateflow`

Jedes dieser Verzeichnisse enthält ein Unterverzeichnis `loesungen`, das wiederum die hier aufgeführten und erläuterten Musterlösungen zum jeweiligen Kapitel enthält. Zusätzlich ist in jedem Lösungsverzeichnis eine *Readme*-Datei vorhanden, in der die Zuordnungen zwischen Aufgabennummer und Lösungsdateien hergestellt werden.

1.2 Änderungen, Aktualisierungen und Internet-Seite

Der gesamte Inhalt der CD-ROM sowie evtl. Aktualisierungen zu Beispielen und Aufgaben, Errata, Links rund um MATLAB und aktuelle Informationen zum Buch sind im Internet verfügbar unter der www-Seite

`www.matlabbuch.de`

Für Anregungen, Kommentare und Mitteilungen über Fehler sind die Autoren den Lesern sehr dankbar.

Dr. Anne Angermann angermann@matlabbuch.de
Dr. Michael Beuschel beuschel@matlabbuch.de
Dr. Martin Rau rau@matlabbuch.de
Ulrich Wohlfarth wohlfarth@matlabbuch.de

2 Buch-Kap. 2: Grundlagen

2.1 Rechengenauigkeit (Buch-Kap. 2.7.1)

Im Gegensatz zur direkten Berechnung (Variable `m`) verstärken sich Fehler durch die begrenzte Genauigkeit der internen Zahlendarstellung bei der iterativen Berechnung (Variable `n`).

```
format long g      % Günstige Zahlendarstellung wählen

n = 1 + 1e-10
m = n^(2^32)       % Direkte Berechnung

for k = 1:32       % Berechnung mit Schleife
    n = n^2;
end
n                  % Ausgabe
```

Ausgabe:

```
n =
              1.0000000001
m =
          1.53648411655459
n =
          1.53648411418528
```

Das Format der Zahlendarstellung lässt sich mit dem Befehl `format` wieder auf die Standardeinstellung zurücksetzen, ist aber auch noch für die nachfolgende Aufgabe nützlich.

2.2 Fibonacci-Folge (Buch-Kap. 2.7.2)

2.2.1 Fibonacci-Folge mit `for`-Schleife

Hinweis: Beim Anfügen neuer Elemente an den bestehenden Zeilenvektor `ausgabe` darf der Audruck `ausgabe(k)+ausgabe(k+1)` keine Leerzeichen enthalten, da diese hier als Trennzeichen zwischen den einzelnen Elementen reserviert sind.

```
ausgabe = [1 1];
for k = 1:10
    ausgabe = [ausgabe ausgabe(k)+ausgabe(k+1)];
end
ausgabe
```

Ausgabe:

```
ausgabe =
     1     1     2     3     5     8    13    21    34    55    89   144
```

2.2.2 Fibonacci-Folge mit `while`-Schleife

Hinweise: Der Wert `end` entspricht immer dem letzten Element eines Vektors, ist also gleichbedeutend mit `length` (*vektor*).

Beim Anfügen neuer Elemente an den bestehenden Zeilenvektor `ausgabe` darf der Audruck `ausgabe(end-1)+ausgabe(end)` keine Leerzeichen enthalten, da diese hier als Trennzeichen zwischen den einzelnen Elementen reserviert sind.

```
ausgabe = [1 1];
while ausgabe (end) < 1e20
    ausgabe = [ausgabe ausgabe(end-1)+ausgabe(end)];
end
ausgabe (end)                % Ausgabe letztes Element (Wert)
length (ausgabe)             % Ausgabe letztes Element (Index)
```

Ausgabe (mit `format long g`):

```
ans =
     1.35301852344707e+020
ans =
    98
```

2.2.3 Fibonacci-Folge direkt

Hinweis: Die verwendete Formel gilt zwar exakt, aber aufgrund der internen Rechenungenauigkeit (hier $< 10^{-12}$) wählt MATLAB ohne Runden eine ungünstige Exponentendarstellung.

```
index = 12:20;
F = (1 + sqrt(5)) / 2;
ausgabe = (F.^index - (1-F).^index) / sqrt(5);   % Elementweise Operation !
round (ausgabe)                                  % Unterdrückung Nachkommastellen
```

Ausgabe:

```
ans =
  Columns 1 through 6
        144         233         377         610         987        1597
  Columns 7 through 9
       2584        4181        6765
```

2.3 Funktion `gerade` (Buch-Kap. 2.7.3)

Bei unendlicher Steigung wird die Funktion mit `return` vorzeitig beendet, da die nachfolgenden Berechnung keine sinnvollen Ergebnisse liefern würden.

Bei falschem Aufruf werden mit `help gerade` die ersten 10 Zeilen als Hilfetext angezeigt.

MATLAB–Funktion gerade.m:

```
%GERADE Geradengleichung aus Koordinaten bestimmen.
%
%   [m,y0] = GERADE(x1,y1,x2,y2) berechnet aus den beiden kartesischen
%   Koordinaten P1 (x1,y1) und P2 (x2,y2) einer Gerade die Parameter
%   m und y0 der Geradengleichung y = m * x + y0.
%
%   [m,y0,r,phi] = GERADE(x1,y1) berechnet aus den kartesischen
%   Koordinaten P1 (x1,y1) die Paramter m und y0 der Geradengleichung
%   y = m * x sowie den Abstand r des Punktes P1 vom Koordinatenursprung
%   und seinen Winkel phi zur x-Achse in Grad.

% gerade.m
% Kapitel 2.7.3 Übungsaufgaben

function [steigung, y0, r, phi] = gerade (x1, y1, x2, y2)

if nargin == 2
    y0     = 0;
    r      = sqrt (x1^2 + y1^2);
    if x1 == 0
        disp ('Steigung unendlich!')
        steigung = sign(y1) * inf;
        phi      = sign(y1) * pi/2 * 180/pi;
        return
    end
    steigung = y1 / x1;
    phi      = sign(y1) * atan2(y1,x1) * 180/pi;

elseif nargin == 4
    if x1 == x2
        disp ('Steigung unendlich!')
        steigung = sign(y2-y1) * inf;
        y0       = [];
        return
    end
    steigung = (y2-y1) / (x2-x1);
    y0       = (x2*y1 - x1*y2) / (x2-x1);
    r        = [];
    phi      = [];

else
    disp ('Falsche Anzahl von Parametern!')
    help gerade
end
```

Beispiel 1:

```
[steigung, y0, r, phi] = gerade (-1, 3)
```

Ausgabe:

```
steigung =
    -3
y0 =
     0
r =
    3.1623
phi =
  108.4349
```

Beispiel 2:

```
[steigung, y0] = gerade (-1, 1, 3, -2)
```

Ausgabe:

```
steigung =
   -0.7500
y0 =
    0.2500
```

2.4 Berechnungszeiten ermitteln (Buch-Kap. 2.7.4)

MATLAB–Funktion `mittelwerte.m`:

```
%MITTELWERTE (X) Berechnungen verschiedener Mittelwert
%   [A,G] = MITTELWERTE (X) berechnet das arithmetische
%   Mittel A und das geometrische Mittel G des Vektors X.

% mittelwerte.m
% Kapitel 2.5 MATLAB-Funktionen
%         Beispiel

function [arithm, geom] = mittelwerte (x)

arithm = mean(x);                     % Arithmetisches Mittel
geom   = prod(x).^(1/length(x));      % Geometrisches Mittel
```

MATLAB–Funktion `mittelwerte2.m`:

```
% mittelwerte2.m
% Kapitel 2.5.1 Funktionen mit variabler Parameterzahl
%         Beispiel

function [out1, out2] = mittelwerte2 (x, schalter, y)

error (nargchk (1, 2, nargin))          % Prüfe Anzahl übergebener Parameter

if nargin == 1
    out1 = mean(x);                     % Arithmetisches Mittel
    out2 = prod(x).^(1/length(x));      % Geometrisches Mittel
elseif nargin == 2
    if schalter == 'g'
        out1 = prod(x).^(1/length(x));  % Geometrisches Mittel
    else
        out1 = mean(x);                 % Arithmetisches Mittel
    end
    out2 = [];                          % Leere Ausgabe für out2
end
```

MATLAB–Funktion `mittelwerte_zeit.m`:

```
% mittelwerte_zeit.m
% Kapitel 2.7.4 Übungsaufgaben

% mittelwerte.m:  Beispiel aus Kapitel 2.5
% mittelwerte2.m: Beispiel aus Kapitel 2.5.1

test   = 1:1000;                        % Testdaten
N      = 10000;                         % Anzahl der Testdurchläufe

for m = 1:10                            % Wiederholen des Tests zum Mitteln der Zeiten
    tic
    for n = 1:N
        [m1, m2] = mittelwerte (test);
    end
    zeit (m) = toc;                     % Zeit für N Durchläufe
    tic
    for n = 1:N
        [m1, m2] = mittelwerte2 (test);
    end
    zeit_2 (m) = toc;                   % Zeit für N Durchläufe
end

zeit_mittelwerte   = mean (zeit)        % Mitteln der Zeiten
zeit_mittelwerte_2 = mean (zeit_2)
```

Der Aufruf von `mittelwerte_zeit` ergibt z. B. folgende Ausgabe:

```
zeit_mittelwerte =
    0.5093
zeit_mittelwerte_2 =
    0.6203
```

3 Buch-Kap. 3: Eingabe und Ausgabe

3.1 Harmonisches Mittel (Buch-Kap. 3.8.1)

MATLAB–Funktion `mittelwerte5.m`:

```
function [arithm, geom, harmon] = mittelwerte (x)

arithm = mean (x);                      % Arithmetisches Mittel
geom = prod (x) ^ (1/length(x));        % Geometrisches Mittel
harmon = length(x) / sum (x.^(-1));     % Harmonisches Mittel
```

MATLAB–Skript (enthalten in `loesungen.m`):

```
clear                          % Initialisieren (alte Werte löschen)

n      = [];                   % Leerer Vektor für Länge der Datenvektoren
arithm = [];                   % Leerer Vektor für Arithmetisches Mittel
geom   = [];                   % Leerer Vektor für Geometrisches Mittel
harmon = [];                   % Leerer Vektor für Harmonisches Mittel
k      = 0;                    % Zähler für bearbeitete Datensätze

disp (' ')
while 1
    x = input ('Geben Sie einen Datenvektor ein: ');
    if length(x)==0,           % Alternativ: if isempty (x),
        break                  % Abbruch bei leerer Eingabe
    end
    n = [n length(x)];         % Länge des Datenvektors anhängen
    k = length(n);             % Zähler für bearbeitete Datensätze
    [arithm(k), geom(k), harmon(k)] = lsg_mittelwerte (x);  % Funktion...
end

disp (' ')
disp (['Es wurden die Mittelwerte für ', num2str(k), ' Datensätze berechnet.'])
disp ('                                Arithmetisch Geometrisch Harmonisch')
disp (' ')
```

Hinweis: Der Befehl `sprintf` benötigt die Daten spaltenweise. Außerdem muss die Anzahl der Ausgabefelder (z. B. `%2d`) mit der Anzahl der übergebenen Spalten übereinstimmen. Andernfalls können Verschiebungen bei der Ausgabe entstehen.

```
ausgabe = sprintf (['Datensatz %2d mit %3d Werten:   ', ...
                    'A = %5.2f    G = %5.2f   H = %5.2f\n'], ...
                    [(1:k); n; arithm; geom; harmon]);
disp (ausgabe)
```

3.2 Einschwingvorgang (Buch-Kap. 3.8.2)

```
zeit = 0:200;                       % Zeit [s]
F    = 0.05;                        % Frequenz [Hz]
T    = 50;                          % Dämpfungszeitkonstante [s]

huellkurve = exp (-zeit/T);
schwingung = cos (2*pi*F*zeit) .* huellkurve;

figure
    plot (zeit, schwingung, 'b-')
    hold on
    plot (zeit, huellkurve, 'r--')
    plot (zeit, -huellkurve, 'g-.')
    plot ([0 T T], [-1 0 -1], 'k:')     % Markierung Zeitkonstante

    xlabel ('Zeit [s]')
    ylabel ('Amplitude')
    title  (' Exponentiell abklingende Schwingung')
    legend ('Schwingung', 'obere Hüllkurve', 'untere Hüllkurve', 'Location', 'NE')
    text   (T, -0.75, '  Zeitkonstante')
```

3.3 Gauß-Glocke (Buch-Kap. 3.8.3)

```
x = -3:0.25:3;                      % x-Vektor (Zeilenvektor)
y = -4:0.25:4;                      % y-Vektor (Spaltenvektor)
[X, Y] = meshgrid (x, y);           % Matrizen der x- und y-Werte

A = X.^2 + Y.^2;                    % Matrix der Abstandsquadrate

sigma = 1;                          % Glättungsfaktor
Z = exp (-A ./ (2*sigma));          % Gauß-Funktion

figure (3)
    surf (X, Y, Z)                  % 3D-Plot
    axis ([-3 3 -4 4 0 1])
    title ('Gauß-Glocke')
figure (4);
    pcolor (X, Y, Z);               % Plot mit pcolor
    axis ([-3 3 -4 4]);
    title ('Gauß-Glocke: pcolor');
figure (5);
    mesh (X, Y, Z)                  % Plot mit mesh
    axis ([-3 3 -4 4 0 1]);
    title ('Gauß-Glocke: mesh')
```

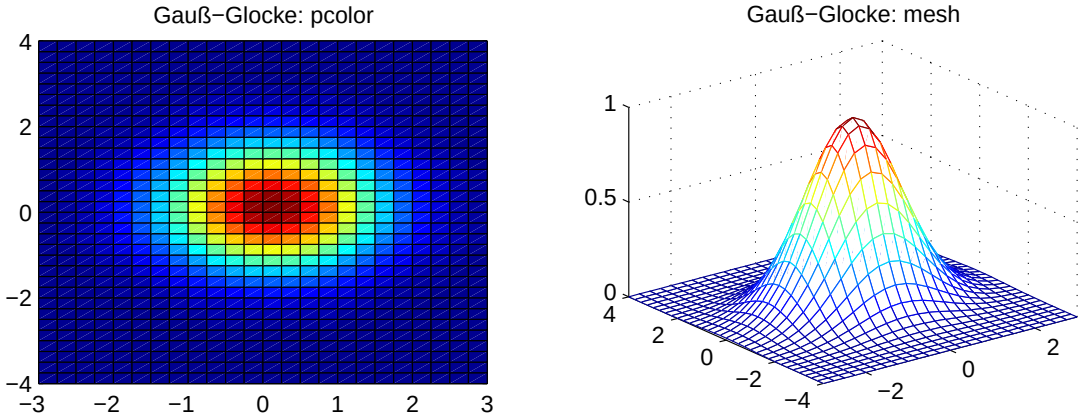


Abb. 3.1: *Gauß-Glocke mit* `pcolor` *und* `mesh`

3.4 Spirale und Doppelhelix (Buch-Kap. 3.8.4)

```
t = 1:0.05:7;                           % Laufvariable
x = cos (2*pi*t) ./ t;
y = sin (2*pi*t) ./ t;
z = log (t);                            % Gleichmäßige Steigung

figure
subplot (121)                           % Spirale
    plot3 (x, y, z)
    view (150, 30)                      % Plot um 180° drehen
    axis ([-1 1 -1 1 0 2])
    box on
    grid off
    title ('Spirale')
```

Für die Doppelhelix wird zusätzlich zur obigen Spirale eine zweite, um 180° gedrehte, Spirale mit negativen x- und y-Koordinaten erzeugt. Der Befehl `surf` füllt die zwischen beiden Spiralen liegenden Flächen aus.

```
subplot (122)                           % Doppelhelix
    surf ([x; -x], [y; -y], [z; z])     % gedrehte Spirale mit -x, -y
    view (150, 30)                      % Plot um 180° drehen
    axis ([-1 1 -1 1 0 2])
    box on
    grid off
    title ('Doppelhelix')
```

3.5 Funktion `geradevek` (Buch-Kap. 3.8.5)

Im ersten Teil der MATLAB–Funktion `geradevek.m` ist der Hilfetext enthalten:

```
function [m,y0,r,phi] = geradevek(x1,y1,x2,y2)
%GERADEVEK Geradengleichung aus Koordinaten-Vektoren bestimmen.
%
%   [m,y0,r,phi] = GERADEVEK(x1,y1) berechnet aus den kartesischen
%   Koordinaten P1 (x1,y1) die Paramter m und y0 der Geradengleichung
%   y = m * x sowie den Abstand r des Punktes P1 vom Koordinatenursprung
%   und seinen Winkel phi zur x-Achse in Grad.
%   Hierbei können x1 und y1 auch Vektoren sein!
%
%   [m,y0] = GERADEVEK(x1,y1,x2,y2) berechnet aus den beiden kartesischen
%   Koordinaten P1 (x1,y1) und P2 (x2,y2) einer Gerade die Parameter
%   m und y0 der Geradengleichung y = m * x + y0.
%   Hierbei können x1, y1, x2 und y2 auch Vektoren sein!

% geradevec.m
% Kapitel 3.8.5 Übungsaufgaben
```

Anschließend folgt die Berechnung der Beispiele, falls die Funktion ohne Übergabeparameter aufgerufen wird:

```
if nargin == 0
    x1 = [1 1];
    y1 = [2 2];
    x2 = [3 2];
    y2 = [1 4];
    disp ('================================================')
    disp ('DEMO geradevek: ')
    disp ('================================================')
    disp (' a) [m,y0] = geradevek(1,2,3,1)')
    [m, y0] = geradevek (x1(1), y1(1), x2(1), y2(1))
    figure
        subplot (331)
        axis ([0 5 -1 5]);
        hold on
        grid on
        title ('a) geradevek(1,2,3,1)')
        plot ([x1(1) x2(1)], [y1(1) y2(1)], 'bo-')
        text (x1(1)+0.5, y1(1)+0.3, ['P_1 = (',num2str(x1(1)),',',num2str(y1(1)),')'])
        text (x2(1),     y2(1)-0.6, ['P_2 = (',num2str(x2(1)),',',num2str(y2(1)),')'])
        text (0.5,       -0.5,      ['m = ',num2str(m),', y0 = ',num2str(y0)])
    disp ('================================================')
    disp (' b) [m,y0] = geradevek([1 1],[2 2],[3 2],[1 4])')
    [m, y0] = geradevek (x1, y1, x2, y2)
        subplot (332)
        axis ([0 5 -1 5])
        hold on
        grid on
        title ('b) geradevek([1 1],[2 2],[3 2],[1 4])')
        plot ([x1(1) x2(1)], [y1(1) y2(1)], 'bo-')
        plot ([x1(2) x2(2)], [y1(2) y2(2)], 'bo-')
        text (x1(1)+0.5, y1(1)+0.3, ['P_1 = P_3 = (',num2str(x1(1)),',',num2str(y1(1)),')'])
        text (x2(1),     y2(1)-0.6, ['P_2 = (',num2str(x2(1)),',',num2str(y2(1)),')'])
        text (x2(2)+0.5, y2(2),     ['P_4 (',num2str(x2(2)),',',num2str(y2(2)),')'])
        text (0.5,       -0.5,      ['m = [',num2str(m(1)),' ',num2str(m(2)),'], ',...
                                    'y0 = [',num2str(y0(1)),' ',num2str(y0(2)),']'])
    disp ('================================================')
    x1 = [0 -2];
    y1 = [4  1];
    disp (' c) [m,y0,r,phi] = geradevek([0 -2],[4 1])')
    [m, y0, r, phi] = geradevek (x1, y1)
        subplot (333)
        axis ([-3 3 -1 5])
        hold on
        grid on
        title ('c) geradevek([0 -2],[4 1])')
        plot ([0 x1(1)], [0 y1(1)], 'bo-')
        plot ([0 x1(2)], [0 y1(2)], 'bo-')
        text (x1(1)+.2, y1(1)+0.2, ['P_1 = (',num2str(x1(1)),',',num2str(y1(1)),')'])
        text (x1(2),    y1(2)+0.4, ['P_2 = (',num2str(x1(2)),',',num2str(y1(2)),')'])
        text (-2.5,     -0.5,      ['m = [',num2str(m(1)),' ',num2str(m(2)),'], ',...
                                    'y0 = [',num2str(y0(1)),' ',num2str(y0(2)),']'])
    disp ('================================================')
```

Hier folgt die Berechnung der Geradengleichungen, falls die Funktion mit 2 Rückgabeparametern aufgerufen wird:

```
elseif nargin == 2
    y0  = zeros (size (x1));
    r   = sqrt (x1.^2 + y1.^2);
    x0  = find (x1==0);              % Indizes der Elemente von x1 gleich Null
    xn0 = find (~x1==0);             % Indizes der Elemente von x1 ungleich Null
    m (xn0)   = y1 (xn0) ./ x1 (xn0);
    phi (xn0) = sign (y1 (xn0)) .* atan2 (y1 (xn0), x1 (xn0)) * 180/pi;
    if x0
        disp (['Steigung m +/- unendlich für Punkt(e) ', num2str(x0),' !'])
        m (x0)    = sign (y1 (x0)) * inf;
        phi (x0)  = sign (y1 (x0)) * 90;
    end
```

Hier folgt die Berechnung der Geradengleichungen und Polarkoordinaten, falls die Funktion mit 4 Rückgabeparametern aufgerufen wird:

```
elseif nargin == 4
    r   = [];
    phi = [];
    x0  = find (x1==x2);             % Indizes der Elemente von x1 gleich x2
    xn0 = find (~(x1==x2));          % Indizes der Elemente von x1 ungleich x2
    m (xn0)  = (y2 (xn0) - y1 (xn0)) ./ (x2 (xn0) - x1 (xn0));
    y0 (xn0) = (x2(xn0).*y1(xn0) - x1(xn0).*y2(xn0)) ./ (x2(xn0)-x1(xn0));
    if x0
        disp (['Steigung m +/- unendlich für Punkt(e) ',num2str(x0),' !'])
        m (x0)   = sign (y2 (x0) - y1 (x0)) * inf;
        y0 (x0)  = 0 * x1 (x0);
    end
```

Bei anderer Anzahl der Übergabeparameter wird eine Fehlermeldung angezeigt und der Hilfetext ausgegeben:

```
else
    disp ('Falsche Anzahl von Parametern!')
    help geradevek
end
```

4 Buch-Kap. 4: Differentialgleichungen in MATLAB

4.1 Feder-Masse Schwinger (Buch-Kap. 4.4.1)

Es wird folgende Substitution durchgeführt: $x_1 = s$, $x_2 = \dot{s}$. Damit ergibt sich das DGL-System:

$$\dot{x}_1 = x_2 \tag{4.1}$$

$$\dot{x}_2 = -\frac{c}{m(t)} \cdot x_1 - \frac{d}{m(t)} \cdot x_2$$

Lösung in MATLAB ohne Verwendung einer Masse-Matrix:

```
% lsg_federmasse.m, Martin Rau
% Masse, Federkonstante, Dämpfung
m0 = 5; c = 150; d = 1;
% Zeit, Anfangswerte
tspan=[0 20]; x0=[0.2; 0];
% Lösung der DGL
[t, x] = ode45(@xpunkt_federmasse,tspan,x0,[],m0,c,d);
% grafische Ausgabe
subplot(211); plot(t,x(:,1),t,x(:,2));
legend('Weg','Geschwindigkeit',4); xlabel('Zeit [s]');
title('Lösungsverlauf'); axis([0 20 -1.2 1.1]); grid on
subplot(212); plot(t, m0-4/20*t);
xlabel('Zeit [s]'); title('Masseverlauf');
axis([0 20 0 5]); grid on
```

```
% xpunkt_federmasse.m, Martin Rau
function dxdt = xpunkt_federmasse(t,x,m0,c,d)
m = m0-4/20*t;
dxdt = [x(2); -c/m*x(1)-d/m*x(2)];
```

Lösung in MATLAB unter Verwendung einer Masse-Matrix:

```
% lsg_federmassemass.m, Martin Rau
% Masse, Federkonstante, Dämpfung
m0 = 5; c = 150; d = 1;
% Zeit, Anfangswerte, Optionen
tspan=[0 20]; x0=[0.2; 0];
options = odeset('Mass', @mass_federmasse);
% Lösung der DGL
[t, x] = ode45(@xpunkt_federmassemass,tspan,x0,options,m0,c,d);
% grafische Ausgabe
subplot(211); plot(t,x(:,1),t,x(:,2));
legend('Weg','Geschwindigkeit',4); xlabel('Zeit [s]');
title('Lösungsverlauf'); axis([0 20 -1.2 1.1]); grid on
subplot(212); plot(t, m0-4/20*t);
xlabel('Zeit [s]'); title('Masseverlauf');
axis([0 20 0 5]); grid on
```

```
% xpunkt_federmassemass.m, Martin Rau
function dxdt = xpunkt_federmasse(t,x,m0,c,d)
dxdt = [x(2); -c*x(1)-d*x(2)];
```

```
% mass_federmasse.m, Martin Rau
function M = mass_federmasse(t,x,m0,c,d)
M = [1 0; 0 m0-4/20*t];
```

Der Lösungsverlauf von Weg, Geschwindigkeit und Masse ist in Abb. 4.1 dargestellt.

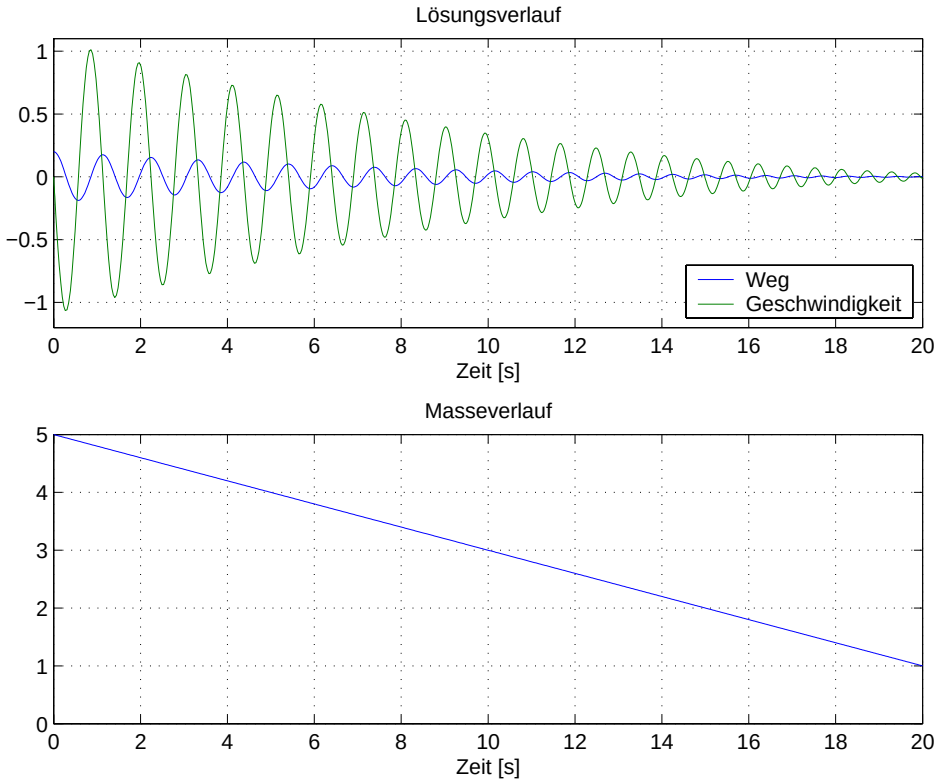


Abb. 4.1: *Lösungsverlauf des Feder-Masse Schwingers*

4.2 Elektrischer Schwingkreis (Buch-Kap. 4.4.2)

Das folgende MATLAB Skript berechnet die geforderte Lösung:

```
% lsg_schwingkreis.m, Martin Rau
% Widerstand, Induktivität, Kapizität
R = 0.5; L = 1e-4; C = 1e-6;
% Zeit, Anfangswerte, Optionen
tend = 1e-3; tspan=[0 tend]; x0=[10; 0];
options = odeset('Events', @event_schwingkreis);
% Lösung der DGL vor dem Umschalten
[tout, xout] = ode45(@xpunkt_schwingkreis,tspan,x0,options,R,L,C);
% ersten Lösungsteil speichern
t = tout; x = xout;
% neue Anfangswerte setzen
tspan = [t(length(t)) tend]; x0 = [x(length(t),1); x(length(t),2)];
% Lösung der DGL nach dem Umschalten
[tout, xout] = ode45(@xpunkt_schwingkreis,tspan,x0,options,R,L,2*C);
% Lösung zusammensetzen
t = [t; tout]; x = [x; xout];
% grafische Ausgabe
subplot(211); plot(t,x(:,1),t,x(:,2));
legend('Spannung U_C','Strom I',4); xlabel('Zeit [s]');
title('Lösungsverlauf'); grid on
```

MATLAB Funktion zur Berechnung der rechten Seite der DGL:

```
% xpunkt_schwingkreis.m, Martin Rau
function dxdt = xpunkt_schwingkreis(t,x,R,L,C)
dxdt = [1/C*x(2); -1/L*x(1)-R/L*x(2)];
```

Ereignisfunktion zur Detektion des Zeitpunkts $t = 0.3 \cdot 10^{-3}$:

```
% event_schwingkreis.m, Martin Rau
function [value, isterminal, direction] = event_schwingkreis(t,x,R,L,C)
value = t-0.3e-3;   % Zeit t= 0.3e-3 detektieren
isterminal = 1; % Integration stoppen
direction = 1; % Nulldurchgang mit positiver Steigung
```

Der Lösungsverlauf von Spannung U_C und Strom I ist in Abb. 4.2 dargestellt. Es ist deutlich zu erkennen, dass sich die Frequenz nach dem Schaltvorgang (Erhöhung der Kapazität) verringert.

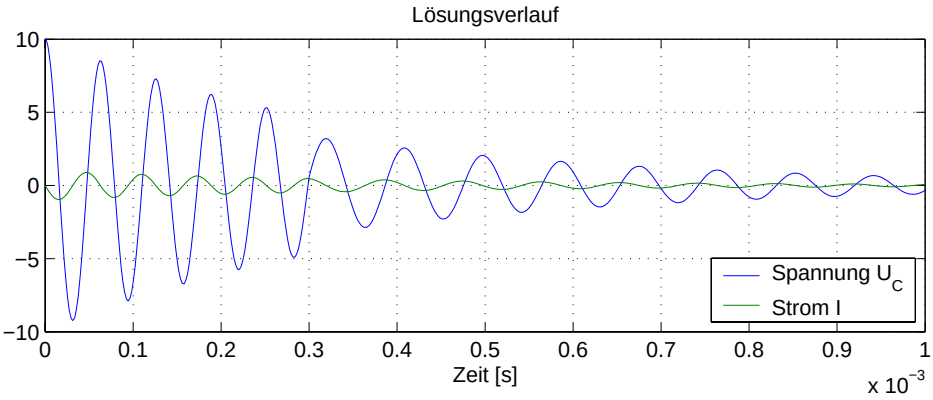


Abb. 4.2: *Lösungsverlauf des elektrischen Schwingkreises*

4.3 Springender Ball (Buch-Kap. 4.4.3)

Zunächst müssen DGL-Systeme erster Ordnung erzeugt werden. Im freien Fall gilt mit der Substitution $x_1 = x$ und $x_2 = \dot{x}$:

$$\dot{x}_1 = x_2 \qquad \dot{x}_2 = -g \tag{4.2}$$

Während der Berührung mit der Oberfläche gilt:

$$\dot{x}_1 = x_2 \qquad \dot{x}_2 = -\frac{c}{m} \cdot x_1 - \frac{d}{m} \cdot x_2 - g \tag{4.3}$$

Die Event Funktion muss Nulldurchgänge in beiden Richtungen erkennen und die Integration anhalten. Sie ist in der Funktion `event_ball` realisiert.

```
% event_ball.m, Martin Rau
function [value, isterminal, direction] = event_schwingkreis(t,x,R,L,C)
value = x(1);   % Nulldurchgang der Position detektieren
isterminal = 1; % Integration stoppen
direction = 0; % Nulldurchgang in beiden Steigungen
```

Die rechten Seiten der Differentialgleichungen sind in den Funktionen `xpunkt_ball_frei` und `xpunkt_ball_elast` in MATLAB codiert.

```
% xpunkt_ball_frei.m, Martin Rau
function dxdt = xpunkt_ball_frei(t,x)
dxdt = [x(2); -9.81];
```

```
% xpunkt_ball_elast.m, Martin Rau
function dxdt = xpunkt_ball_elast(t,x)
m = 1; c = 300; d = 0.5; g = 9.81;
dxdt = [x(2); -c/m*x(1)-d/m*x(2)-g];
```

In den folgenden MATLAB Kommandos wird die Lösung bestimmt. Es wird jeweils eine Phase freier Fall gelöst, dann eine Phase der Berührung mit der Oberfläche und dies wird so lange wiederholt, bis die Endzeit erreicht ist. Die Lösungsabschnitte werden aneinandergesetzt und bei jedem Übergang wird als neuer Anfangswert der Endwert der letzten Lösungsphase verwendet.

```
% lsg_ball.m, Martin Rau
% Zeit, Anfangswerte, Optionen
tend = 20; tspan=[0 tend]; x0=[1; 0];
options = odeset('Events', @event_ball);
% Variablen zur Kummulation der Lösung
t = []; x = [];
% Schleife zur Lösungsbestimmung
while 1
  % Lösung freier Fall
  [tout, xout] = ode45(@xpunkt_ball_frei,tspan,x0,options);
  % Lösung zusammensetzen
  t = [t; tout]; x = [x; xout];
  if tout(length(tout))>=tend
    break
  end
  % neue Anfangswerte, Zeitspanne
  tspan = [tout(length(tout)) tend]; x0 = [0 xout(length(tout),2)];
  % Lösung elastischer Stoß
  [tout, xout] = ode45(@xpunkt_ball_elast,tspan,x0,options);
  % Lösung zusammensetzen
  t = [t; tout]; x = [x; xout];
  if tout(length(tout))>=tend
    break
  end
  % neue Anfangswerte, Zeitspanne
  tspan = [tout(length(tout)) tend]; x0 = [0 xout(length(tout),2)];
end
% grafische Ausgabe
subplot(211); plot(t,x(:,1),t,x(:,2));
legend('Position','Geschwindigkeit',4); xlabel('Zeit [s]');
title('Lösungsverlauf'); grid on
```

Abb. 4.3 zeigt den Lösungsverlauf für eine Zeitspanne von 20 s. Die Phasen "freier Fall" und "Berührung mit der Oberfläche" sind deutlich unterscheidbar.

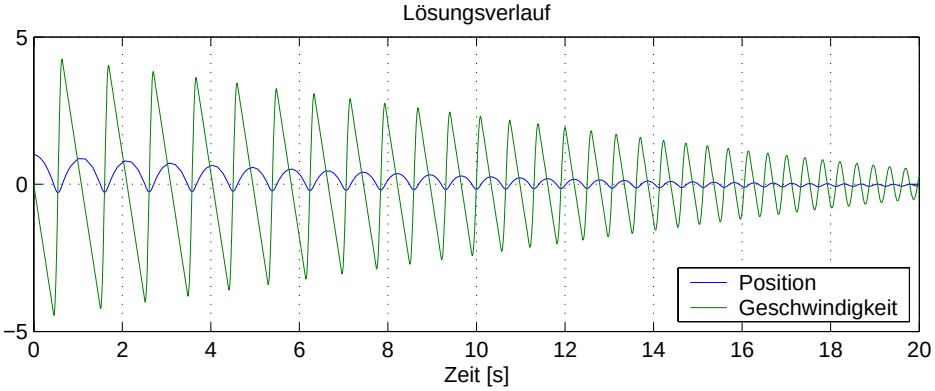


Abb. 4.3: *Lösungsverlauf des springenden Balls*

4.4 Kettenlinie (Buch-Kap. 4.4.4)

Zunächst wird ein DGL-System erster Ordnung durch die Substitution $y_1 = y$ und $y_2 = y'$ erzeugt. Dieses lautet

$$y_1' = y_2 \qquad y_2' = a \cdot \sqrt{1 + y'^2} \tag{4.4}$$

Die rechte Seite von Gleichung (4.4) wird in der Funktion `ystrich_kette` implementiert. Der bekannte Parameter a wird vom Solver an die Funktionen für die rechte Seite der DGL und die Residuen der Randwerte übergeben.

```
% ystrich_kette.m, Martin Rau
function dydx = ystrich_kette(x,y,a)
dydx = [y(2); a*sqrt(1+y(2)^2)];
```

Die Berechnung der Residuen erfolgt in der Funktion `bc_kette`. Auch hier wird der Parameter a vom Solver übergeben.

```
% bc_kette.m, Martin Rau
function res = bc_kette(ya,yb,a)
res = [ya(1); yb(1)];
```

Die Lösung des Randwertproblems erfolgt im MATLAB-Skript `bvp_kette`. Als Schätzwert für die Startlösung wird eine Gerade $y = -0.5$ verwendet und diese in zwischen $x = 0$ und $x = 1$ mit 50 Punkten gerastert. Die Struktur für die Startlösung wird mittels `bvpinit` erzeugt. Im Aufruf des Solvers werden keine Optionen benutzt, aber der Parameter a an `ystrich_kette` und `bc_kette` übergeben.

```
% bvp_kette.m, Martin Rau
% Daten des Seils
a = 4;
% Schätzwert für Startlösung
xmesh = linspace(0,1,50);
yini(1) = -0.5; yini(2) = 0;
solinit = bvpinit(xmesh,yini);
% Lösung berechnen
sol = bvp4c(@ystrich_kette,@bc_kette,solinit,[],a)
% Ergebnis grafisch darstellen
subplot(211); plot(sol.x,sol.y(1,:))
```

Die Lösung der DGL der Kettenlinie mit den angegebenen Daten ist in Abb. 4.4 dargestellt.

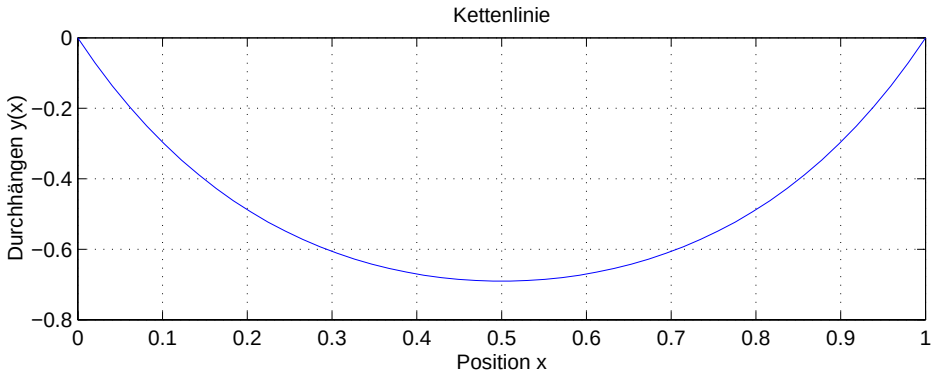


Abb. 4.4: *Lösung des Randwertproblems zur Kettenlinie*

5 Buch-Kap. 5: Regelungstechnische Funktionen

Readme-Datei

```
% readme.txt
% Dateien zur Lösung der Aufgaben aus Kapitel 5.6:
% Regelungstechnische Funktionen - Control System Toolbox

Datei            Kap.    Erläuterung
================================================================================
control_lsg.m            Skript zum Aufrufen aller Lösungsdateien

control_lsg1.m   5.6.1   Erstellen von LTI-Modellen
control_lsg2.m   5.6.2   Verzögerte Übertragungsglieder
control_lsg3.m   5.6.3   Verzögerte Übertragungsglieder zeitdiskretisiert
control_lsg4.m   5.6.4   Typumwandlung
control_lsg5.m   5.6.5   Stabilitätsanalyse
control_lsg6.m   5.6.6   Regelung der stabilen PT_2-Übertragungsfunktion
control_lsg7.m   5.6.7   Regelung der instabilen PT_2-Übertragungsfunktion
control_lsg8.m   5.6.8   Kondition und numerische Instabilität

figure_defaults.m  Grundeinstellungen für Figure-Plots (Linienstärke etc.)
```

5.1 Erstellen von LTI–Modellen (Buch-Kap. 5.6.1)

Lösung-Datei: `control_lsg1.m`

Wie in der Lösungsdatei leicht nachzuvollziehen, werden zuerst die 5 Übertragungsfunktionen als einzelne TF-LTI-Modelle definiert und der Eigenschaft `Notes` jeweils der entsprechende Name zugewiesen.

```
P   = tf (1,1)              % P-Glied
I   = tf (1,[1 0])          % I-Glied
D   = tf ([1 0],1)          % D-Glied
PI  = tf ([1 0.1],[1 0])    % PI-Glied
PD  = tf ([1 1],1)          % PD-Glied

P.Notes   = 'P-Glied' ;
I.Notes   = 'I-Glied' ;
D.Notes   = 'D-Glied' ;
PI.Notes  = 'PI-Glied' ;
PD.Notes  = 'PD-Glied' ;
```

Im zweiten Schritt wird nun ein LTI–Modell `sys1` erstellt, daß 5 Eingänge und einen Ausgang besitzt. Ebenso wird ein Cell Array erzeugt, der die Namen der einzelnen Modelle enthält.

```
% Systeme erzeugen
sys1     = [P,I,D,PI,PD];
sys1name = [P.Notes,I.Notes,D.Notes,PI.Notes,PD.Notes];
```

Diese Definitionen dienen nur zur Automatisierung der Ausgabe in einer Figure, wie sie die folgenden Zeilen zeigen:

```
figure
sys1m    = max(size(sys1)) ;
for m = 1:sys1m,
    subplot(sys1m,4,4*m-3)
        axis([0 100 0 100])
        text(0,40,[sys1name(m)])
        axis off;
    subplot(sys1m,4,4*m-2)
        if isproper(sys1(m)) , step(sys1(m)) , else , axis off; end;
    subplot(sys1m,4,4*m-1)
        bode(sys1(m))
    subplot(sys1m,4,4*m)
        nyquist(sys1(m))
end; % m
```

Nach dem Öffnen einer leeren Figure wird die größer Dimension des LTI–Modells `sys1` ermittelt und der Variablen `sys1m` zugewiesen, die die Anzahl der untereinanderstehenden Subplots bestimmt (hier fünf, für jede Übertragungsfunktion eine). In der FOR–Schleife werden dann für diese fünf Systeme jeweils nebeneinander 4 Subplots erzeugt, in die folgendes geplottet wird:

- Eine nicht sichtbare Achsenumgebung (`axis off`), in die mit der Laufvariablen `m` der Name der Übertragungsfunktion aus `sys1name` ausgelesen und angzeigt wird.
- Sprungantwort, wobei MATLAB die Sprungantwort nicht berechnen kann für LTI-Modelle, bei denen der Rang des Zählerpolynoms größer als der Rang des Nennerpolynoms ist, die also global differenzierendes Verhalten haben. Dies muss mit dem Befehl `isproper` überprüft werden.
- Bode-Diagramm
- Nyquist-Diagramm

Da die Nummerierung der Subplot aufsteigend von links nach rechts und dann von oben nach unten erfolgt, wird die richtige Nummer anhand der Laufvariablen `m` automatisch ermittelt, abhängig von der Spalte für den jeweiligen Subplot, z. B. `4*m-3=4*m-(4-1)` für die Spalte 1.

Die nächste Zeile dient lediglich für die Ausgabe der Figure in eine farbige EPS-Graphikdatei (`-depsc`) nahmens `lti_modell.eps`, falls die Variable `lpr` existiert und den Wert 1 hat. die Überprüfung der Existenz

```
if (exist('lpr')~=0)&(lpr==1) , print('-depsc',['lti_modell.eps']) , end
```

5.2 Verzögerte Übertragungsglieder (Buch-Kap. 5.6.2)

Lösung-Datei: `control_lsg2.m`

Das vorgehen entspricht dem in der Lösung zu Aufgabe 5.6.1, es müssen lediglich andere Systeme und andere Systemnamen angegeben werden. Hierzu werden zusätzliche Variablen `T`, `V`, `wn` und `D` erzeugt:

```
T  = 0.2 ; V  = 0.8 ;
wn = T/0.005 ; D = 0.05 ;

PT1 = tf(V,[T 1])                  % PT1-Glied
PT2 = tf(V*wn^2,[1 2*D*wn wn^2])   % PT2-Glied
IT1 = tf(1,[1 0])*PT1              % IT1-Glied
DT1 = tf([1 0],1)*PT1              % DT1-Glied
Tt  = tf(1,1,'IOdelay',0.1)        % Totzeit-Glied
```

Das IT_1- und DT_1-Glied lassen sich leicht als Verknüpfung eines I- bzw. D-Gliedes mit dem vorher definierten PT_1-Glied erzeugen, das Totzeitglied durch Setzen der Eigenschaft `IOdelay`.

Das Definieren des Systems `sys2` und des Cell Arrays `sys2name` sowie die Ausgabe–Schleife entsprechen der Lösung von Aufgabe 5.6.1.

5.3 Verzögerte Übertragungsglieder zeitdiskretisiert (Buch-Kap. 5.6.3)

Lösung-Datei: `control_lsg3.m`

Die Zeitdiskretisierung geschieht hier einfach mit dem Befehl `c2d` und durch die Angabe der Abtastzeit `Ts`.

```
% Diskrete Systeme erzeugen
Ts = 0.1 ;
sys2d = c2d([PT1,PT2,IT1,DT1,Tt],Ts);
```

Ausgegeben werden hier nur die Systemnamen, die Sprungantwort und das Bode-Diagramm, es sind also nur 3 Spalten mit Subplots. Ansonsten gestaltet sich die Lösung wie die von Aufgabe 5.6.1.

5.4 Typumwandlung (Buch-Kap. 5.6.4)

Lösung-Datei: `control_lsg4.m`

PT_1–Übertragungsfunktion:

Wird die PT_1–Übertragungsfunktion

$$x \;=\; \frac{V}{1+s\,T}\cdot u \qquad \text{und} \qquad y = x$$

in die zugrundeliegende Differentialgleichung umgerechnet, so lautet diese:

$$\dot{x} \;=\; -\frac{1}{T}\cdot x \;+\; \frac{1}{T}\cdot V\cdot u$$

Hieraus können sofort die – hier skalarwertigen – Systemmatrizen abgelesen werden:

$$\mathbf{A} \;=\; -\frac{1}{T}\,; \qquad \mathbf{B} \;=\; \frac{V}{T}\,; \qquad \mathbf{C} \;=\; 1\,; \qquad \mathbf{D} \;=\; 0$$

In MATLAB geschieht die Umwandlung einfach mit:

```
PT1ss = ss(PT1)
```

Werden diese Systemmatrizen nun in Gl. (5.73) eingesetzt, so lässt sich schnell die Übereinstimmung der beiden Darstellungsarten in MATLAB zeigen:

$$\mathbf{C}\cdot\left(\,s\mathbf{E}-\mathbf{A}\,\right)^{-1}\cdot\mathbf{B} \;=\; \frac{\mathbf{C}\cdot\mathbf{B}}{s-\mathbf{A}} \;=\; \frac{\mathbf{C}\cdot\mathbf{B}}{-\mathbf{A}}\cdot\frac{1}{-\mathbf{A}^{-1}s+1} \;\stackrel{!}{=}\; \frac{V}{1+sT}$$

$$\Rightarrow \qquad T \;=\; -\mathbf{A}^{-1}\,, \quad V \;=\; \frac{\mathbf{C}\cdot\mathbf{B}}{-\mathbf{A}}$$

In MATLAB lässt sich das schnell mit den folgenden Zeilen überprüfen:

```
[ V T ; PT1ss.c*PT1ss.b/-PT1ss.a -PT1ss.a^-1 ]
```

PT_2–Übertragungsfunktion:

Für die PT_2–Übertragungsfunktion

$$x \;=\; \frac{V\omega_0^2}{s^2+s\,2D\omega_0+\omega_0^2}\cdot u \qquad \text{und} \qquad y = x$$

wird ebenso vorgegangen, wobei hier allerdings die Differentialgleichung 2. Ordnung

$$\ddot{x}+\;2D\omega_0\cdot\dot{x}\;+\;\omega_0^2\cdot x \;=\; V\omega_0^2\cdot u$$

in ein System aus 2 Differentialgleichungen 1. Ordnung umgewandelt werden muss. Hierzu wird x durch x_2 substituiert und zusätzlich $\dot{x}_2 = x_1$ eingeführt. Es folgt damit:

$$\dot{x}_1 = -2D\omega_0 \cdot x_1 - \omega_0^2 \cdot x_2 + V\omega_0^2 \cdot u$$

$$\dot{x}_2 = x_1$$

Dies lässt sich in Matrixschreibweise mit $\mathbf{x} = [x_1\ x_2]^T$ darstellen als

$$\dot{\mathbf{x}} = \begin{bmatrix} -2D\omega_0 & -\omega_0^2 \\ 1 & 0 \end{bmatrix} \cdot \mathbf{x} + \begin{bmatrix} V\omega_0^2 \\ 0 \end{bmatrix} \cdot u \qquad \text{und} \qquad y = \begin{bmatrix} 0\ 1 \end{bmatrix} \cdot \mathbf{x}$$

Hieraus können sofort die Systemmatrizen abgelesen werden:

$$\mathbf{A} = \begin{bmatrix} -2D\omega_0 & -\omega_0^2 \\ 1 & 0 \end{bmatrix}; \qquad \mathbf{B} = \begin{bmatrix} V\omega_0^2 \\ 0 \end{bmatrix}; \qquad \mathbf{C} = \begin{bmatrix} 0\ 1 \end{bmatrix}; \qquad \mathbf{D} = 0$$

In MATLAB geschieht die Umwandlung einfach mit

```
PT2ss = ss(PT2)
```

Werden diese Systemmatrizen nun in Gl. (5.73) eingesetzt, so ergibt sich rein formal:

$$\mathbf{C} \cdot (s\mathbf{E} - \mathbf{A})^{-1} \cdot \mathbf{B} = \begin{bmatrix} 0\ c_2 \end{bmatrix} \cdot \frac{1}{\det(s\mathbf{E} - \mathbf{A})} \cdot \begin{bmatrix} s - a_{22} & -a_{12} \\ a_{21} & s - a_{11} \end{bmatrix} \cdot \begin{bmatrix} b_1 \\ 0 \end{bmatrix}$$

Unter Ausnutzung der beiden Null–Elemente in $\mathbf{C}$ und $\mathbf{B}$ folgt:

$$\mathbf{C} \cdot (s\mathbf{E} - \mathbf{A})^{-1} \cdot \mathbf{B} = \frac{c_2 \cdot a_{21} \cdot b_1}{s^2 - s(a_{11} + a_{22}) + (a_{11}a_{22} - a_{12}a_{21})} \stackrel{!}{=} \frac{V\omega_0^2}{s^2 - s\,2D\omega_0 + \omega_0^2}$$

Der Koeffizientenvergleich in MATLAB ergibt erwartungsgemäß:

```
[ PT2ss.c(2)*PT2ss.a(2,1)*PT2ss.b(1) -PT2ss.a(1,1)-PT2ss.a(2,2) det(PT2ss.a) ; ...
V*wn^2 2*D*wn wn^2 ]
```

5.5 Stabilitätsanalyse (Buch-Kap. 5.6.5)

Lösung-Datei: `control_lsg5.m`

1. **Bestimmung der Art der Eigenwerte abhängig von a, b und c:**

   Die 3 Parameter werden als Vektoren definiert, anschließend werden in 3 Schleifen jeweils die Polstellen des Systems `sys` ermittelt und über den Befehl `disp` die Werte für a, b und c sowie die Anzahl der Realteile kleiner, gleich und größer Null und die Existenz konjungiert–komplexer Nullstellen angezeigt.

```
a = [ -1 1 ]; b = [ -1 0 1 ]; c = [ -1 0 1 ]; K = 1 ;

disp(sprintf('  a  |  b  |  c  || Re<0 | Re=0 | Re>0 || konj |'))
disp('-------------------------------------------------')
for l = 1:1:length(a)
  for m = 1:1:length(b)
    for n = 1:1:length(c)
      sys = ss([ -b(m)/a(l) -c(n)/a(l) ; 1 0 ],[K/a(l) 0]',[0 1],0) ;
      p = pole(sys) ;
      disp(sprintf(' %2d  | %2d  | %2d  ||  %2d  |  %2d  |  %2d  ||  %2d  |',...
            a(l),b(m),c(n),sum(real(p<0)),...
            sum(real(p==0)),sum(real(p>0)),sum(imag(p)~=0)/2 ))
    end % m
  end % l
  disp('-------------------------------------------------')
end % k
```

2. **Plotten des Polverhaltes für Variation von c zwischen -1 und $+1$**

   Zuerst werden wieder die Parameter gesetzt, wobei hier laut Angabe nur `c` als Vektor definiert wird. Abgespeichert werden die Polstellen getrennt nach Real- und Imaginärteil in den beiden Matrizen `Pre` und `Pim`, wobei diese rekursiv erweitert werden.

```
a = 1 ; b = 1 ; c = [-1:0.05:1] ; K = 1 ;
Pre = [] ; Pim = [] ;

for n=1:1:length(c)
    sys = ss([ -b/a -c(n)/a ; 1 0 ],[K/a 0]',[0 1],0) ;
    p = pole(sys) ;
    Pre = [ Pre , real(p) ] ; Pim = [ Pim , imag(p) ] ;
end
```

3. **Plot erzeugen:**

   Schlussendlich wird die Figure mit den folgenden Befehlen erzeugt:

```
figure
    hold on
    title(['Polstellen (a=',num2str(a),', b=',num2str(b),'; o Anfang, x Ende)'])
    xlabel('Realteil'); ylabel('Imaginärteil')
    text(-1.8,0.8,'Stabil'); text( 0.2,0.8,'INSTABIL')

    plot(Pre(1,:),Pim(1,:),'r-')
    plot(Pre(2,:),Pim(2,:),'b--')
    plot(Pre(1,1),Pim(1,1),'ro',Pre(1,n),Pim(1,n),'rx')
    plot(Pre(2,1),Pim(2,1),'bo',Pre(2,n),Pim(2,n),'bx')
    ax = axis ;
    plot([0 0],[ax(3) ax(4)],'k-')
```

5.6 Regelung der stabilen PT_2-Übertragungsfunktion (Buch-Kap. 5.6.6)

Lösung-Datei: `control_lsg6.m`

1. **Streckenverhalten:**

   Nachdem die PT_2-Übertragungsfunktion mit den folgenden Zeilen definiert wurde

```
% Strecke definieren
a = 1 ; b = 0.5 ; c = 1 ; K = 1 ;
sys = ss([ -b/a -c/a ; 1 0 ],[K/a ; 0],[0 1],0)
```

   kann das Systemverhalten z. B. mit den folgenden Befehlen untersucht werden

```
% 1. Streckenverhalten
pole(sys)
[p,z] = pzmap(sys)
dcgain(sys)
[wn,z] = damp(sys)

figure
    subplot(221) , step(sys)
    subplot(222) , impulse(sys)
    subplot(223) , bode(sys)
    subplot(223) , pzmap(sys)

ltiview(sys)
```

2. **Regelung:**

   Zur Regelung wird das Regler-System `reg` mit den geforderten Charakteristika erzeugt und dann mittels des Befehls `feedback` der geschlossene Regelkreis berechnet. Mit `pole` wird überprüft, ob dieser stabil ist und mit `dcgain` die Gleichverstärkung ausgegeben.

```
P = tf([10],[1])                          % P-Regler
sys_P = feedback(P*tf(sys),1);            % Geregeltes System
pole(sys_P)
dcgain(sys_P)

PI = tf([10 1],[1 0])                     % PI-Regler
sys_PI = feedback(PI*tf(sys),1);          % Geregeltes System
pole(sys_PI)
dcgain(sys_PI)

PID = tf([5 10 1],[1 0])                  % PID-Regler
sys_PID = feedback(PID*tf(sys),1);        % Geregeltes System
pole(sys_PID)
dcgain(sys_PID)
```

Anschließend wird in eine Figure das Ergebnis der Sprungantworten des geschlossenen und in eine zweite Figure die Phasen- und Amplitudenränder des offenen Regelkreises geplottet:

```
figure
    subplot(311) , step(sys,sys_P,20)   , legend('Strecke','P-Regler',4)
    subplot(312) , step(sys,sys_PI,20)  , legend('Strecke','PI-Regler',4)
    subplot(313) , step(sys,sys_PID,20) , legend('Strecke','PID-Regler',4)

figure
    subplot(311) , margin(P*tf(sys))   , legend('P-Regler',3)
    subplot(312) , margin(PI*tf(sys))  , legend('PI-Regler',3)
    subplot(313) , margin(PID*tf(sys)) , legend('PID-Regler',3)
```

5.7 Regelung der instabilen PT_2-Übertragungsfunktion (Buch-Kap. 5.6.7)

Lösung-Datei: `control_lsg7.m`

1. **Regelung mit P-Regler**

   Wird die Übertragungsfunktion des geschlossenen Regelkreises aufgestellt,

   $$G(s) \;=\; \frac{G_V}{1+G_V G_R} \;=\; \frac{Z_V N_R}{N_V N_R + Z_V Z_R} \;=\; \frac{Z_V}{N_V + Z_V} \;=\; \frac{k \cdot V_p}{a \cdot s^2 + b \cdot s + c + k \cdot V_p}$$

   mit G_V, G_R Übertragungsfunktionen im Vorwärts- bzw. Rückwärtszweig, Z_V, N_V Zähler- und Nennerpolynom der Übertragungsfunktionen im Vorwärtszweig und Z_R, N_R Zähler- und Nennerpolynom der Übertragungsfunktionen im Rückwärtszweig (hier beide 1) sowie V_P Verstärkungsfaktor des Reglers, so erhält man für das Nennerpolynom folgende Gleichung:

   $$N(s) \;=\; N_V + Z_V \;=\; a \cdot s^2 + b \cdot s + c + k \cdot V_p$$

   Die Lösung des charakteristische Polynoms lautet dann:

   $$\lambda_{1,2} \quad = \quad -\frac{b}{2a} \;\pm\; \frac{1}{2a} \cdot \sqrt{b^2 - 4a(c + k \cdot V_p)}$$

   Mit der Stabilitätstabelle 5.1 erhält man schnell die Lösung für die Stabilität des Regelkreise in Abhängigkeit von c und k, die dann den Verstärkungsfaktor des Reglers V_P bestimmen.

   $$c + k \cdot V_p \begin{cases} < \; 0 & \rightarrow \quad \text{ein Re}(\chi) > 0\text{, Regelkreis instabil} \\ = \; 0 & \rightarrow \quad \text{ein Re}(\chi) = 0\text{, Regelkreis grenzstabil} \\ > \; 0 & \rightarrow \quad \text{beide Re}(\chi) < 0\text{, Regelkreis stabil} \end{cases} \quad \Longrightarrow \quad V_P \begin{cases} < \; -\frac{c}{k} \\ = \; -\frac{c}{k} \\ > \; -\frac{c}{k} \end{cases}$$

   Wie leicht zu zeigen, läßt sich das System zwar durch den P-Regler stabilisieren, jedoch ist die stationäre Gleichverstärkung immer unterschiedlich von 1 für k, c und V_P ungleich Null.

Programmiertechnisch wird wieder wie in Aufgabe 5.6.6 vorgegangen:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 1. Instabiles System (c=-1)

% Strecke definieren
a = 1 ; b = 0.5 ; c = -1 ; K = 1 ;
sys = ss([ -b/a -c/a ; 1 0 ],[K/a ; 0],[0 1],0)

% P-Regler: -c/K < Vp =>  stabil
reg = tf([-c/K+10*abs(c/K)],[1])                % Regler-Koeffizient
sys_r = feedback(reg*tf(sys),1);                % Geregeltes System
```

2. **Regelung mit Zustandsregler mit Sollwertanpassung**

   Für den Zustandsregler mit Sollwertanpassung wird zuerst mittels `place` die Rückführmatrix `K` bestimmt und dann der Vorfaktor `Kv` nach der angegebenen Formel bestimmt. Mit `pole(sys_z)` kann überprüft werden, ob die Systempole mit den Wunschpolen übereinstimmen. Anschließend werden die Sprungantworten des geregelten Systems mit P-Regler, Zustandsregler und Zustandsregler mit Sollwertanpassung verglichen.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% 2. Instabiles System (c=-1)

% Zustands-Regler mit Sollwertanpassung
K  = place(sys.a,sys.b,[-1.1 -1]')              % Regler-Rückführmatrix
Kv = -1 / (sys.c*inv(sys.a-sys.b*K)*sys.b)      % Sollwertanpassung
sys_z = ss(sys.a-sys.b*K,sys.b,sys.c,0);        % Geregeltes System
p_z = pole(sys_z)

figure
    step(sys_r,'g:',sys_z,'b--',sys_z*Kv,'b-')
    legend('P-Regler','Zustands-Regler','Sollwertanpassung',4)
```

3. **Regelung mit Zustandsregler mit Führungsintegrator**

   Statt dem negativen c wird nun b negativ, die Strecke kann nicht mehr durch einen P-Regler stabilisiert werden, es gibt immer mindestens eine positive Reelle Nullstelle des charakteristischen Polynoms!

   Nach der erneuten Streckendefinition wird der Zustandsregler mit Sollwertanpassung entworfen wie der vorangehenden Aufgabe, zusätzlich erhält die Zustandsdarstellung der geregelten Strecke einen zweiten Eingang für die Störgröße z.

```
% Zustands-Regler mit Sollwertanpassung
K  = place(sys.a,sys.b,[-1.1 -1]')              % Regler-Rückführmatrix
Kv = -1 / (sys.c*inv(sys.a-sys.b*K)*sys.b)      % Sollwertanpassung
sys_s = ss(sys.a-sys.b*K,[sys.b [2;0]],sys.c,0)*Kv;
```

   Für den Zustandsregler mit Führungsintegrator muß noch der zusätzlich Zustand x_I in die Zustandsbeschreibung aufgenommen werden. Die geregelte Strecke `sys_zf` erhält dann auch noch den zusätzlichen Eingang für die Störgröße.

```
% Zustands-Regler mit Führungsintegrator
sys_f = ss([sys.a [0;0] ; -sys.c 0],[sys.b ; 0],[sys.c 0],0)
Kf = place(sys_f.a,sys_f.b,[-1.1 -1 -0.9])      % Regler-Rückführmatrix
sys_zf = ss(sys_f.a-sys_f.b*Kf,[0 2 ; 0 0 ; 1 0],[sys.c 0],0)
```

   Die Zeitsignale sowie die Ausgabe in eine Figure werden wie folgt erzeugt:

```
% Zeit und Eingangssignale erzeugen
t = 0:0.01:50 ;
w = t > 0 ;                                     % Führungs-Sprung bei t=1
z = 0.3*(t > 15) - 0.5*(t > 35) ;               % Stör-Sprung bei t=15
y_s  = lsim(sys_s,[w' z'],t) ;                  % Sollwertanpassung
y_zf = lsim(sys_zf,[w' z'],t) ;                 % Führungsintegrator

figure
    hold on; grid on
    plot(t,z,'k-.',t,y_s,'b--',t,y_zf,'r-')
    legend('Störung','Sollwertanpassung','Führungsintegrator',4)
```

5.8 Kondition und numerische Instabilität (Buch-Kap. 5.6.8)

Lösung-Datei: `control_lsg8.m`

1. **Kondition (Natürliche Instabilität):**

   Wie in der Angabe verlangt, wird in einer Schleife jeweils die Werte von `a` geändert und die Lösung des ungestörten `x` und des gestörten`xe` linearen Gleichungssystems ermittelt. Ausgegeben werden dann der Schleifenzähler `n`, `a`, die Konditionszahl von `A` und log10(cond(A)) sowie der Lösungsvektor `x` und `xe`.

```
format long;
b = [ 1 ; 0 ]
E = [ 0.0001 0.0001 ; 0.0001 -0.0001 ] ;
for n = 1:1:9 ,
    a = 1 + 10^-n ;
    A = [ 1 a ; a 1 ] ;
    x  = A \ b ;
    xe = (A+E) \ b ;
    disp('==================')
    n
    a
    cond(A)
    log10(cond(A))
    x'
    xe'
end
```

2. **Numerische Instabilität**

   Hier wird die gleiche Schleife wie bei der Konditionsaufgabe durchlaufen, nur wird hier die Summe $x_1 + x_2$ des Lösungsvektors gebildet, und zwar zum einen als Summe der Elemente des Lösungsvektors des linearen Gleichungssystems und zum anderen mit der (mathematisch korrekten) Formel $1/(1+a)$.

```
b = [ 1 ; 0 ]
for n = 1:1:9 ,
    a = 1 + 10^-n ;
    A = [ 1 a ; a 1 ] ;
    x  = A \ b ;
    disp('===============')
    n
    a
    [ sum(x) ; 1/(1+a) ]
end
```

6 Buch-Kap. 6: Signalverarbeitung

6.1 Signaltransformation im Frequenzbereich (Buch-Kap. 6.5.1)

Zunächst wird das Ausgangssignal (Rechteck) erzeugt und mittels DFT transformiert:

```
t = 0:0.001:0.0499;                 % Zeitvektor   1000 Hz
x = square (2*pi*20*t);             % Rechtecksignal 20 Hz

y = fft (x);                        % DFT (ohne Normierung)
n = length (y);                     % Länge des Datensatzes
f = (0:n-1) / n / diff(t(1:2));     % Frequenzvektor zum Plotten
```

Hier werden die Fourierkoeffizienten skaliert:

```
                                    % Erzeugt Skalierungsvektor
skal = imag (exp (j*pi/2*(0:n-1)));   % 0 1 0 -1   0 1   0 -1   ...
skal = skal ./ [1 (1:n-1)];           % 0 1 0 -1/3 0 1/5 0 -1/7 ...

y_skal = y .* skal;                 % Skalierte Fourierkoeffizienten
x_skal = ifft (y_skal);             % inverse DFT (ohne Normierung)
```

Ausgabe:

```
figure
subplot (121)
    plot (t, x, 'b--')
    hold on
    plot (t, real (x_skal), 'r-')
    axis ([0 max(t) -1.2 1.2])
    legend ('Rechteck', 'Dreieck', 'Location', 'NE')
    title ('Zeitsignale')
    xlabel ('Zeit [s]')

subplot (122)
    stem (f, -2*imag (y), 'bx')
    hold on
    stem (f, -2*imag (y_skal), 'ro')
    axis ([0 max(f)/2 -10 70])
    legend ('Rechteck', 'Dreieck', 'Location', 'NE')
    title ('Fourierkoeffizienten b_k')
    xlabel ('Frequenz [Hz]')
```

6.2 Signalanalyse und digitale Filterung (Buch-Kap. 6.5.2)

Der Datensatz wurde mit den folgenden Zeilen erzeugt und gespeichert:

```
t = 0.001:0.001:1;                                % Zeitvektor Fs = 1000 Hz
xx = 12 * sin (2*pi*4*t) + ...                    % Sollsignal      4 Hz
      4 * sin (2*pi*12*t);                        %                12 Hz
x =  xx + ...
     20 * cos (2*pi*63*t) + ...                   % Störsignal     63 Hz
     18 * cos (2*pi*137*t) .* sin (2*pi*29*t) +  ... % 137-29 Hz, 137+29 Hz
      4 + randn (size(t));                        % Rauschen

datei = fopen ('ueb_signal.dat', 'w');            % Signal speichern (ASCII)
fprintf (datei, '%5.3f  %6.2f\r\n', [t' x']');    % \r\n = CRLF im DOS-Format
fclose (datei);

save ueb_signal t x                               % Signal speichern (MAT)
```

Hinweis: Die Multiplikation eines 137-Hz mit einem 29-Hz-Signal (d. h. Amplitudenmodulation) entspricht einer Faltung im Frequenzbereich. Dadurch werden Anteile mit 108 und 166 Hz erzeugt ($= 137 \pm 29$ Hz).

Lösung der Aufgabe (siehe Abb. 6.1):

```
load ueb_signal.dat                           % Signal laden
t = ueb_signal (:,1)';                        % Zeitvektor
x = ueb_signal (:,2)';                        % Messdaten-Vektor

T = diff (t(1:2));                            % Abtastzeit
N = length (x);                               % Länge des Datenvektors
f = [0:floor((N-1)/2)] / (N*T);               % Frequenzvektor für Plot

X = fft (x);                                  % Fouriertransformation
X = [X(1) 2*X(2:floor((N-1)/2)+1)] / N;       % Begrenzung und Normierung

                                              % FIR-Filter auslegen
ord    = 40;                                  % Ordnung des FIR-Filters
B      = fir1 (ord, 0.04);                    % FIR-Filter berechnen f_g = 20 Hz
x_filt = filter (B, 1, x);                    % Daten filtern
x_filt = x_filt - mean (x_filt);              % Gleichanteil ausblenden

figure
subplot (221)
    plot (t, x, 'b-')
    xlabel ('Zeit [s]')
    title ('Ungefiltertes Signal')
subplot (222)
    stem (f, abs (X), 'bo')
    axis ([0 500 0 25])
    xlabel ('Frequenz [Hz]')
    title ('Spektrum')
subplot (223)
    plot (t, x_filt, 'r-')
    xlabel ('Zeit [s]')
    title ('Gefiltertes Signal')
```

Die Übertragungsfunktion des FIR-Filters kann folgendermaßen berechnet und angezeigt werden:

```
F   = [-ord/2:0.1:ord/2] / (ord*T);
amp = zeros (size (F));
for i = -ord/2:ord/2
   amp = amp + B(i+ord/2+1) * cos (2*pi*F*i*T);
end
subplot (224)
    plot (F, abs (amp))
    axis ([0 500 0 1])
    xlabel ('Frequenz [Hz]')
    title ('Filter-Übertragungsfunktion')
```

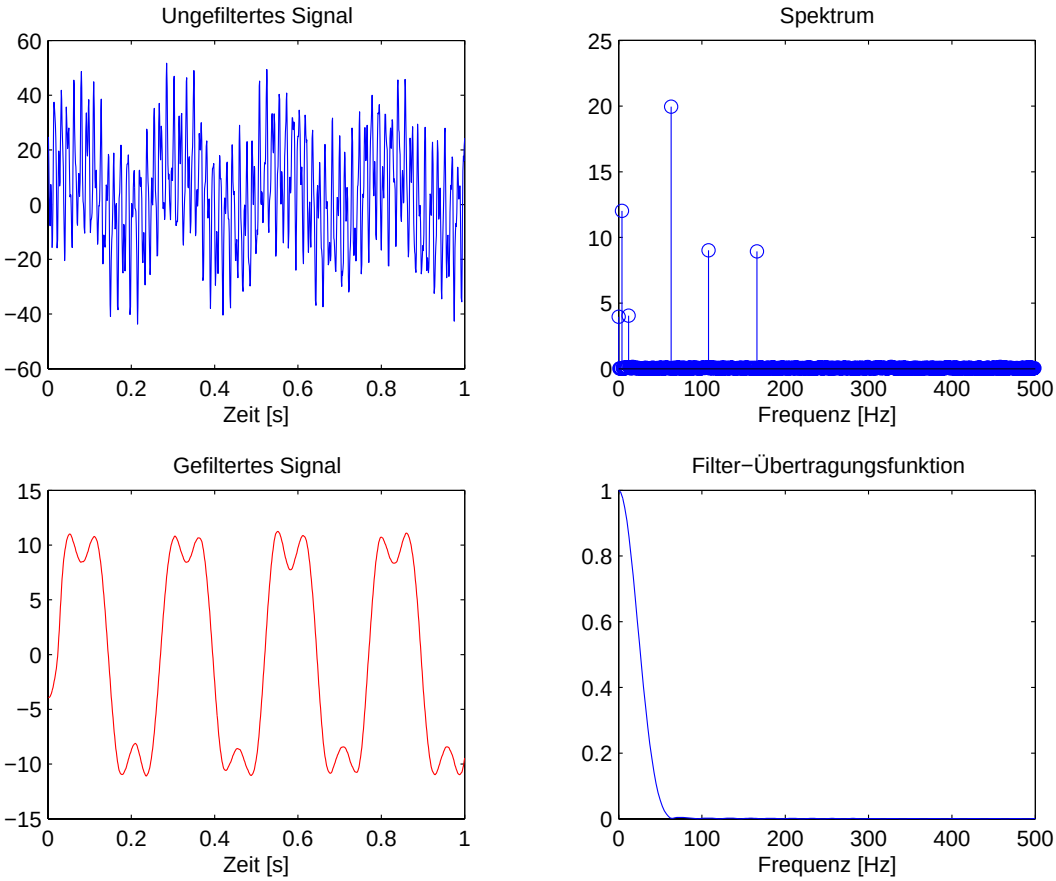


Abb. 6.1: *Signale und Spektren der Lösung zu Buch-Kap. 6.5.2*

6.3 Analoger Bandpass (Buch-Kap. 6.5.3)

Es werden ein Bandpass (10 . . . 100 Hz) und eine Bandsperre (20 . . . 70 Hz) erzeugt. Die Faltung beider Übertragungsfunktionen ergibt das gewünschte Filter.

```
[B1, A1] = butter (12, 2*pi*[10 100], 's');        % Bandpass 10...100 Hz
[B2, A2] = butter (12, 2*pi*[20 70], 'stop', 's'); % Bandsperre 20...70 Hz

A = conv (A1, A2);                                 % Nenner Falten
B = conv (B1, B2);                                 % Zähler Falten

W = 2*pi * linspace (0, 300, 301);                 % Frequenzvektor [rad/s]
H = freqs (B, A, W);                               % Gesamt-Übertragungsfunktion

figure
    plot ([0 10 10 20 20 70 70 100 100 150], ...       % ideale Übertragungsfunktion
          [0  0  1  1  0  0  1   1   0   0], 'k--')
    hold on
    plot (W/2/pi, abs (H), 'b-')
    axis ([0 150 0 1.2])
    xlabel ('Frequenz [Hz]')
    title ('Gesamt-Filter')
    legend ('Ideal', '8. Ordnung')
```

6.4 Digitaler IIR-Bandpass (Buch-Kap. 6.5.4)

6.4.1 Bandpass mit `butter`

Die einfachste Berechnung erfolgt mit dem Befehl `butter` analog zur vorherigen Aufgabe. Die Grenzfrequenzen sind jetzt normiert; `freqz` benötigt zusätzlich die Abtastfrequenz *Fs*.

```
Fs = 300;                                          % Abtastfrequenz 300 Hz
[B1, A1] = butter (8, [10 100]/(Fs/2));            % Bandpass 10...100 Hz
[B2, A2] = butter (8, [20 70]/(Fs/2), 'stop');     % Bandsperre 20...70 Hz

A = conv (A1, A2);                                 % Nenner Falten
B = conv (B1, B2);                                 % Zähler Falten

F = linspace (0, 150, 301);                        % Frequenzvektor [Hz]
H = freqz (B, A, F, Fs);                           % Gesamt-Übertragungsfunktion

    plot (F, abs (H), 'g-')
```

6.4.2 Bandpass mit Prototyp-Tiefpass

Die Berechnung mittels Prototyp-Tiefpass führt zum selben Ergebnis (siehe Abb. 6.2), erfordert aber ein individuelles Pre-warping für jede Grenzfrequenz. Die Übertragungsfunktion des Filters im Laplace-Bereich kann nicht direkt transformiert werden, da dann das Pre-warping nur für eine Frequenz optimal passen würde.

Liegt die Abtastfrequenz deutlich über der höchsten Grenzfrequenz, nimmt die Bedeutung des Pre-warping aber ab und die direkte Tranfsormation liefert eine gute Näherung; allerdings kann es dann zu numerischen Problemen aufgrund der starken Größenunterschiede der Koeffizienten geben.

```
[z, p, k]    = buttap (8);                         % Prototyp-Tiefpass
[a, b, c, d] = zp2ss (z, p, k);                    % Umrechnung in Zustandsform

Fs = 300;                                          % Abtastfrequenz 300 Hz
omega        = [10 100] * 2 * pi;                  % Grenzfrequenz
omega        = 2 * Fs * tan (omega / Fs / 2);      % Grenzfrequenz mit Pre-warping
[e, f, g, h] = lp2bp (a, b, c, d, sqrt(prod(omega)), diff(omega));   % Bandpass
[r, s, t, u] = bilinear (e, f, g, h, Fs);          % Dig. Filter mit Pre-warping
[B1, A1]     = ss2tf (r, s, t, u);                 % Übertragungsfunktion

omega        = [20 70] * 2 * pi;                   % Grenzfrequenz
omega        = 2 * Fs * tan (omega / Fs / 2);      % Grenzfrequenz mit Pre-warping
[e, f, g, h] = lp2bs (a, b, c, d, sqrt(prod(omega)), diff(omega));   % Bandsperre
[r, s, t, u] = bilinear (e, f, g, h, Fs);          % Dig. Filter mit Pre-warping
[B2, A2]     = ss2tf (r, s, t, u);                 % Übertragungsfunktion

A = conv (A1, A2);                                 % Nenner Falten
B = conv (B1, B2);                                 % Zähler Falten

F = linspace (0, 150, 301);                        % Frequenzvektor [Hz]
H = freqz (B, A, F, Fs);                           % Gesamt-Übertragungsfunktion

    plot (F, abs (H), 'c--')
```

6.4.3 Bandpass mit `yulewalk`

Der Befehl `yulewalk` erzeugt ein Filter mit höherer Flankensteilheit aber auch deutlich höherer Welligkeit sowohl in den Durchlass- als in den auch Sperrbereichen.

```
Fs = 300;                                          % Abtastfrequenz 300 Hz
frequenz  = [0 10 10 20 20 70 70 100 100 150] / (Fs/2); % Normierte Frequenzen
amplitude = [0  0  1  1  0  0  1   1   0   0];     % Amplituden
[B, A] = yulewalk (16, frequenz, amplitude);       % IIR-Filter mit Ordnung 2*8

F = linspace (0, 150, 301);                        % Frequenzvektor [Hz]
H = freqz (B, A, F, Fs);                           % Gesamt-Übertragungsfunktion

    plot (F, abs (H), 'r-')
    axis ([0 150 0 1.2])
    xlabel ('Frequenz [Hz]')
    title ('Gesamt-Filter')
    legend ('Ideal', 'Analog', 'IIR', 'IIR Prototyp', 'YULEWALK')
```

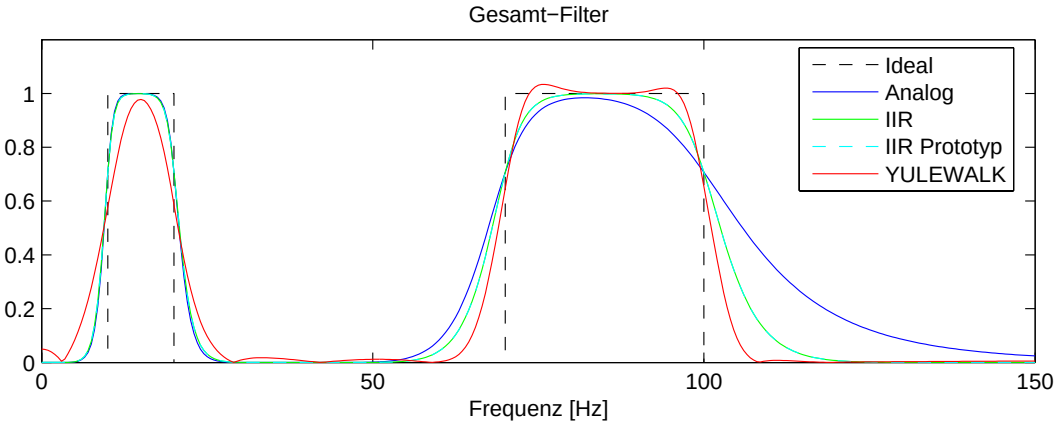


Abb. 6.2: *Übertragungsfunktionen der Lösung zu Buch-Kap. 6.5.4*

7 Buch-Kap. 7: Optimierung

7.1 Nullstellenbestimmung (Buch-Kap. 7.8.1)

Funktionsdefinition:

```
% f_nullst.m
% Funktion zu Aufgabe 7.8.1, Optimization TB
function f = f_nullst(x)
f = exp(-x.^2)-x.^2+x;
```

Bestimmung der Nullstellen:

```
% nullst_aufg.m
% Lösung zu Aufgabe 7.8.1, optimization TB
% Nullstellenbestimmung

format compact; clear all
% f_nullst.m definiert die Funktion f(x)=exp(-x^2)-x^2+x

ezplot(@f_nullst,[-1 2]);  % Funktion graphisch darstellen
grid on
% 2 Nullstellen: in der Nähe von x=-0.5 und x=1
x01 = [-1 0];  % erstes Suchintervall
x02 = [1 2];   % zweites Suchintervall
nullst1 = fzero(@f_nullst,x01)
nullst2 = fzero(@f_nullst,x02)
```

7.2 Lösen von Gleichungssystemen (Buch-Kap. 7.8.2)

1. Lösung des lineares Gleichungssystems:

```
% glsys_aufg.m
% Lösung zu Aufgabe 7.8.2, optimization TB
% Lösen von Gleichungssystemen

format compact; clear all

% Lineares Gleichungssystem
A = [3 -4 5; -3 4 1; 5 -1 -3];
b = [1; 4; 0];
x = A \ b
```

2. Funktionsdefinition für das nichtlineare Gleichungssystem:

```
% f_glsys.m
% Aufgabe 7.8.2.2, Optimization TB
function f = f_glsys(x)
f = [x(1)-exp(-x(2)^2), x(2)+exp(-x(1))];
```

Funktionsdefinition für das nichtlineare Gleichungssystem mit Rückgabe der Jacobi-Matrix:

```
% J_glsys.m
% Aufgabe 7.8.2.2, Optimization TB
function [f, J] = J_glsys(x)
f = [x(1)-exp(-x(2)^2), x(2)+exp(-x(1))];
J = [1, 2*x(2)*exp(-x(2)^2); -exp(-x(1)), 1];
```

Lösung des nichtlinearen Gleichungssystems:

```
% glsys_aufg.m
% Lösung zu Aufgabe 7.8.2, optimization TB
% Lösen von Gleichungssystemen

% Nichtlineares Gleichungssystem
% In der Funktion f_glsys.m ist die Funktion
% f_1(x)=x(1)-exp(-x(2)^2), f_2(x)=x(2)+exp(-x(1)) definiert

% Optionen und Startwert
options=optimset(@fsolve);
options=optimset(options,'Display','iter');
x0 = [0 0];
% Gleichungssystem lösen ohne Jacobi-Matrix
[x,fval,exitflag]=fsolve(@f_glsys,x0,options)

% In der Funktion J_glsys.m wird zusätzlich zum Funktionswert
% die Jacobi-Matrix berechnet
% Optionen und Startwert
options=optimset(@fsolve);
options=optimset(options,'Display','iter','Jacobian','on');
x0 = [0 0];
% Gleichungssystem lösen mit Jacobi-Matrix
[x,fval,exitflag]=fsolve(@J_glsys,x0,options)
```

7.3 Minimierung ohne Nebenbedingungen (Buch-Kap. 7.8.3)

1. Funktionsdefinition zur Minimumbestimmung:

```
% f_minonbmin_aufg1.m
% Funktion zu Aufgabe 7.8.3.1, Optimization TB
function f = f_minonbmin_aufg1(x)
f = -x.^2-5*exp(-x.^2);
```

Funktionsdefinition zur Maximumbestimmung:

```
% f_minonbmax_aufg1.m
% Funktion zu Aufgabe 7.8.3.1, Optimization TB
function f = f_minonbmax_aufg1(x)
f = x.^2+5*exp(-x.^2);
```

Bestimmung der Extrema:

```
% minonb_aufg1.m
% Lösung zu Aufgabe 7.8.3.1, optimization TB
% Minimierung ohne Nebenbedingungen
format compact; clear all

ezplot(@f_minonbmin_aufg1,[-2.5 2.5]);  % Funktion graphisch darstellen
grid on
% 1 Minimum in der Nähe von x=0
% 2 Maxima in der Nähe von x=+-1.25
% In der Funktion f_minonbmin_aufg1.m wird die Funktion f(x)=-x.^2-5*exp(-x.^2)
% zur Minimumsuche definiert, in der Funktion f_minonbmax_aufg1.m wird
% die Funktion f(x)=-1*f_minonbmin_aufg1 zur Maximumsuche definiert

% Optionen setzen
options = optimset(@fminbnd); options=optimset(options,'Display','iter');
% Minimum bestimmen
x1 = -0.5; x2 = 0.5;   % Bereich einschränken
[xmin,ymin] = fminbnd(@f_minonbmin_aufg1,x1,x2,options)
% Maxima bestimmen
x1 = -2; x2 = 0.5;   % Bereich einschränken
[xmax1,ymax1] = fminbnd(@f_minonbmax_aufg1,x1,x2,options)
x1 = 0.5; x2 = 2;   % Bereich einschränken
[xmax2,ymax2] = fminbnd(@f_minonbmax_aufg1,x1,x2,options)
```

2. Funktionsdefinition zur Minimumbestimmung:

```
% f_minonbmin_aufg2.m
% Funktion zu Aufgabe 7.8.3.2, Optimization TB
function f = f_minonbmin_aufg2(x)
f = 10.*exp(-x(1).^2-x(2).^2).*(x(1).^3+x(2).^2+0.5);
```

Funktionsdefinition zur Maximumbestimmung:

```
% f_minonbmax_aufg2.m
% Funktion zu Aufgabe 7.8.3.2, Optimization TB
function f = f_minonbmax_aufg2(x)
f = -1.*(10.*exp(-x(1).^2-x(2).^2).*(x(1).^3+x(2).^2+0.5));
```

Bestimmung der Extremwerte ohne Nebenbedingungen:

```
% minonb_aufg2.m
% Lösung zu Aufgabe 7.8.3.2, optimization TB
% Minimierung ohne Nebenbedingungen
format compact; clear all

% Funktion graphisch darstellen
[X,Y] = meshgrid(-3:0.1:3,-3:0.1:3);
Z = 10.*exp(-X.^2-Y.^2).*(X.^3+Y.^2+0.5);
mesh(X,Y,Z)
xlabel('x_1'); ylabel('x_2')
axis([-3 3 -3 3 -4 4])
grid on
% Das Minimum liegt in der Nähe von x_1=-1, x_2=0
% Die Maxima liegen in der Nähe von x_1=0, x_2=1; x_1=0, x_2=-1
% und x_1=1.5, x_2=0
% In der Funktion f_minonbmin_aufg2.m wird die Funktion
% f(x)=10.*exp(-x(1).^2-x(2).^2).*(x(1).^3+x(2).^2) definiert
% In der Funktion f_minonbmax_aufg2.m wird -1*f(x) definiert
% Optionen setzen
options = optimset(@fminunc);
options=optimset(options,'Display','iter','LargeScale','off');

% Minimum bestimmen
x0 = [-1 0];          % Startwert vorgeben
[xmin,ymin] = fminunc(@f_minonbmin_aufg2,x0,options)
% Maxima bestimmen
x0 = [0 1];          % Startwert vorgeben
[xmax1,ymax1] = fminunc(@f_minonbmax_aufg2,x0,options)
x0 = [0 -1];          % Startwert vorgeben
[xmax2,ymax2] = fminunc(@f_minonbmax_aufg2,x0,options)
x0 = [1.25 0];          % Startwert vorgeben
[xmax3,ymax3] = fminunc(@f_minonbmax_aufg2,x0,options)
```

7.4 Minimierung unter Nebenbedingungen (Buch-Kap. 7.8.4)

Funktionsdefinition der Zielfunktion:

```
% f_minmnb.m
% Funktion zu Aufgabe 7.8.4, Optimization TB
function f = f_minmnb(x)
f = 10*exp(-x(1).^2-x(2).^2)*(x(1).^3+x(2).^2+0.5);
```

Funktionsdefinition der Gleichungsnebenbedingung:

```
% neb_minmnb.m
% Gleichungsnebenbedingung zu Aufgabe 7.8.4, Optimization TB
function [c, ceq] = neb_minmnb(x)
c = [];
ceq = x(1)^2/1.25^2 + (x(2)+1)^2/4 -1;
```

Die ungefähre Lage des globalen Minimums kann graphisch abgelesen werden und der Startwert für die Optimierung wird in dessen Nähe gewählt. Bestimmung des globalen Minimums:

```
% minmnb_aufg.m
% Lösung zu Aufgabe 7.8.4, optimization TB
% Minimierung unter Nebenbedingungen
format compact; clear all

% In der Funktion f_minmnb.m ist die Funktion
% f(x)=10*exp(-x(1)^2-x(2)^2)*(x(1)^3+x(2)^2+0.5) definiert
% Funktion f_minmnb.m graphisch darstellen
[X,Y] = meshgrid(-3:0.1:3,-3:0.1:3);
Z = 10.*exp(-X.^2-Y.^2).*(X.^3+Y.^2+0.5);
mesh(X,Y,Z)
xlabel('x_1'); ylabel('x_2')
axis([-3 3 -3 3 -4 4])
grid on

% Nebenbedingung in selben Plot einzeichnen
hold on
t = 0:0.1:2*pi;
x = 1.25*sin(t); y = 2*cos(t)-1;
for i=1:1:length(t)
   z(i) = f_minmnb([x(i) y(i)]);
end
plot3(x,y,z)

% Das Minimum liegt in der Nähe von x_1=-1, x_2=0
% Optionen setzen
options = optimset(@fmincon);
options=optimset(options,'Display','iter','LargeScale','off');
% In der Funktion neb_minmnb.m ist die Gleichungs-Nebenbedingung definiert

% Minimum bestimmen
x0 = [-1 0];         % Startwert vorgeben
[xmin,ymin] = fmincon(@f_minmnb,x0,[],[],[],[],[],[],@neb_minmnb,options)
```

7.5 Ausgleichspolynom (Buch-Kap. 7.8.5)

Berechnung der Polynomkoeffizienten mit \ (`slash`) und `polyfit`:

```
% ausglpoly_aufg.m
% Lösung zu Aufgabe 7.8.5, optimization TB
% Berechnung eines Ausgleichspolynoms
format compact; clear all

% Datensatz laden
load ausgl_daten.mat

% Polynom erster Ordnung
C = [ones(length(u),1) u];   % Matrizen erstellen
d = y;
x1 = C \ d                   % Lösung berechnen

% Polynom dritter Ordnung
C = [ones(length(u),1) u u.^2 u.^3];   % Matrizen erstellen
d = y;
x3 = C \ d                             % Lösung berechnen

% Polynom fünfter Ordnung
C = [ones(length(u),1) u u.^2 u.^3 u.^4 u.^5]; % Matrizen erstellen
d = y;
x5 = C \ d                                     % Lösung berechnen

% Lösung mit polyfit
x1 = polyfit(u,y,1)
x3 = polyfit(u,y,3)
x5 = polyfit(u,y,5)
```

Ein Polynom dritter Ordnung ist ausreichend, da sich die Polynomkoeffizienten bei Erhöhung der Ordnung auf fünf praktisch nicht mehr ändern. Ferner sind die Koeffizienten vierter und fünfter Ordnung gleich Null.

7.6 Curve Fitting (Buch-Kap. 7.8.6)

Definition der Ausgleichsfunktion:

```
% f_curvefit.m
% Funktion zu Aufgabe 7.8.6, Optimization TB
function f = f_curvefit(x,xdata)
f = exp(-xdata./x(1)).*(sin(x(2).*xdata-x(4))+sin(x(3).*xdata-x(5)));
```

Bestimmung der Funktionsparameter:

```
% curvefit_aufg.m
% Lösung zu Aufgabe 7.8.6, optimization TB
% Nichtlineares Curve Fitting
format compact; clear all

% Datensatz laden
load curve_daten.mat

% Optionen setzen
options = optimset(@lsqcurvefit);
options = optimset(options,'LargeScale','off','Display','iter');

% Parameter bestimmen
x0=[2 4 10 1 2];    % Startwerte vorgeben
x = lsqcurvefit(@f_curvefit,x0,t,y,[],[],options)
```

7.7 Lineare Programmierung (Buch-Kap. 7.8.7)

Der gesuchte Vektor der Startzeitpunkte wird definiert zu:

$$\mathbf{t} = [t_1\ t_2\ t_3\ t_4\ t_5\ t_6\ t_7]^T \tag{7.1}$$

Die Zielfunktion ist:

$$f(\mathbf{t}) = \mathbf{g}^T\mathbf{t} \qquad \text{mit} \qquad \mathbf{g} = [\,0\,0\,0\,0\,0\,0\,1\,]^T \tag{7.2}$$

Alle Startzeitpunkte müssen positiv sein, d.h. es ergeben sich folgende Ungleichungsnebenbedingungen:

$$t_i \geq 0 \qquad \Leftrightarrow \qquad -t_i \leq 0 \tag{7.3}$$

Die Nebenbedingungen werden in der Matrixschreibweise $\mathbf{A}_1 \cdot \mathbf{t} \leq \mathbf{b}_1$ dargestellt. Die Matrix $\mathbf{A}_1$ und der Vektor $\mathbf{b}_1$ ergeben sich zu:

$$\mathbf{A}_1 = \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix} \qquad \mathbf{b}_1 = [\,0\,0\,0\,0\,0\,0\,0\,] \tag{7.4}$$

Desweiteren muss beachtet werden, dass ein Arbeitsschritt erst beginnen kann, wenn die vorgelagerten Arbeitsschritte bereits beendet sind. Aus dem Netzplan in Abb. 7.10 im Buch ergeben sich die folgenden Ungleichungsnebenbedingungen:

$$t_2 - t_1 \geq d_1 \qquad \Leftrightarrow \qquad t_1 - t_2 \leq -d_1 \tag{7.5}$$
$$t_3 - t_1 \geq d_1 \qquad \Leftrightarrow \qquad t_1 - t_3 \leq -d_1 \tag{7.6}$$
$$t_4 - t_3 \geq d_3 \qquad \Leftrightarrow \qquad t_3 - t_4 \leq -d_3 \tag{7.7}$$
$$t_5 - t_3 \geq d_3 \qquad \Leftrightarrow \qquad t_3 - t_5 \leq -d_3 \tag{7.8}$$
$$t_6 - t_2 \geq d_2 \qquad \Leftrightarrow \qquad t_2 - t_6 \leq -d_2 \tag{7.9}$$
$$t_6 - t_4 \geq d_4 \qquad \Leftrightarrow \qquad t_4 - t_6 \leq -d_4 \tag{7.10}$$
$$t_6 - t_5 \geq d_5 \qquad \Leftrightarrow \qquad t_5 - t_6 \leq -d_5 \tag{7.11}$$
$$t_7 - t_6 \geq d_6 \qquad \Leftrightarrow \qquad t_6 - t_7 \leq -d_6 \tag{7.12}$$

Auch diese Nebenbedingungen werden in der Matrixschreibweise $\mathbf{A}_2 \cdot \mathbf{t} \leq \mathbf{b}_2$ dargestellt. Die Matrix $\mathbf{A}_2$ und der Vektor $\mathbf{b}_2$ ergeben sich zu:

$$\mathbf{A}_2 = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix} \qquad \mathbf{b}_2 = [\, -d_1 \; -d_1 \; -d_3 \; -d_3 \; -d_2 \; -d_4 \; -d_5 \; -d_6 \,]^T \tag{7.13}$$

Die Matrix $\mathbf{A}$ und der Vektor $\mathbf{b}$ als Argumente des Befehls `linprog` werden aus den Matrizen und Vektoren $\mathbf{A}_1$, $\mathbf{A}_2$, $\mathbf{b}_1$ und $\mathbf{b}_2$ zusammengesetzt.

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_1 \\ \mathbf{A}_2 \end{bmatrix} \qquad \mathbf{b} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix} \tag{7.14}$$

Die Lösung des optimalen Terminplans wird im folgenden MATLAB-Skript realisiert.

```
% netzplan.m
% Aufgabe 7.8.7
% Bestimmung der optimalen Startzeitpunkte eines Produktionsprozesses

% Vektor g der Zielfunktion
g = [zeros(1,6), 1];

% Dauern der Arbeitsschritte
d1 = 0; d2 = 2; d3 = 3; d4 = 3; d5 = 2; d6 = 4; d7 = 0;

% Matrix A der Ungleichungsnebenbedingungen
A = [-1.*eye(7);
     1 -1 0 0 0 0 0;
     1 0 -1 0 0 0 0;
     0 0 1 -1 0 0 0;
     0 0 1 0 -1 0 0;
     0 1 0 0 0 -1 0;
     0 0 0 1 0 -1 0;
     0 0 0 0 1 -1 0;
     0 0 0 0 0 1 -1];
b = [zeros(1,7) -d1 -d1 -d3 -d3 -d2 -d4 -d5 -d6].';

% optimale Startzeitpunkte bestimmen
[x,fval,exitflag] = linprog(g,A,b)

Optimization terminated successfully.
x =
    0.0000
    2.3914
    0.0000
    3.0000
    3.4028
    6.0000
   10.0000
fval =
   10.0000
exitflag =
     1
```

Die Lösung ist nur eindeutig hinsichtlich des sogenannten kritischen Pfads des Produktionsprozesses. Dieser verläuft von a_1 über a_3, a_4 und a_6 nach a_7. Alle anderen Anfangszeitpunkte können sich in gewissen Grenzen bewegen, so dass sich eine mehrdeutige Lösung ergibt. Die nicht ganzzahligen Anfangszeitpunkte der Lösung sind solche mehrdeutigen Lösungen.

8 Buch-Kap. 8: Simulink Grundlagen

8.1 Nichtlineare Differentialgleichungen (Buch-Kap. 8.10.1)

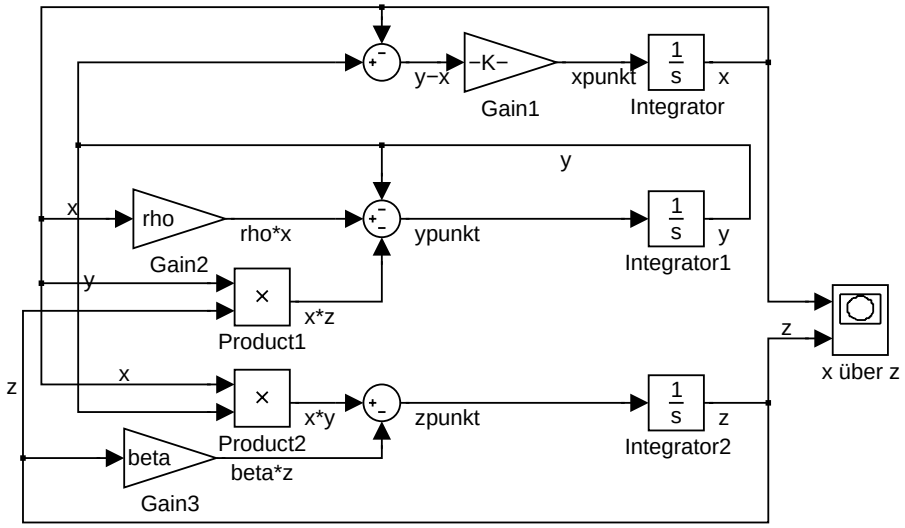


Abb. 8.1: *Simulink–Modell* `lorenz_attractor.mdl`

Im Simulink–Modell `lorenz_attractor.mdl` werden die Differentialgleichungen des Lorenz Attraktors nachgebildet. Die Parameter σ, ρ und β werden im MATLAB-Script `lorenz_attractor_ini.m` definiert und als Callback-Routine *Model initialization function* im *Callbacks*-Register des *Model Properties* Editor eingetragen. Im Register *Data Import/Export* der *Configuration Parameters* Dialogbox werden die Simulationszeit als *tout* und die Zustandsvariablen als *xout* auf den Workspace gespeichert. Im MATLAB–Script `lorenz_attractor_plot.m` (Callback-Routine *Simulation stop function*) können diese dann graphisch, z.B. mit dem Befehl `plot3` dargestellt werden.

8.2 Gravitationspendel (Buch-Kap. 8.10.2)

Da es sich bei der DGL des reibungsbehafteten Gravitationspendels

$$\ddot{\varphi} = -\frac{g}{l} \cdot \sin\varphi \; - \; \frac{1}{m\,l} \; \cdot \; 1.3 \cdot arctan(50 \cdot \dot{\varphi})$$

um eine Differentialgleichung 2. Ordnung handelt, werden 2 *Integrator*–Blöcke benötigt.

In beiden Integratoren wird durch das Setzen von *Initial condition source* auf *external* ein zusätzlicher Eingang erzeugt. An diese werden die mit Hilfe von *Constant*–Blöcken erzeugten Signale für φ_0 und $\dot{\varphi}_0$ angelegt. Da in der Maske des Subsystems φ_0 in ° eingegeben wird, muss entweder bereits im *Constant*–Block oder gleich anschließend z. B. mit Hilfe eines *Gain* die Umrechnung in die Einheit rad/sec erfolgen. Für die Ausgabe im *Scope* wird wieder in ° umgerechnet. Die Signale φ, $\dot{\varphi}$ und $\ddot{\varphi}$ müssen, um auf der obersten Modellebene aufgezeichnet werden zu können, im Subsystem auf *Outport*–Blöcke geführt werden.

Die nichtlineare Rückführung des Winkels kann z. B. mit Hilfe eines *Trigonometric Function*-Blocks, die Reibungskennlinie direkt mit einem *Fcn*-Block oder auch mit Hilfe eines *Trigonometric Function*-Blocks programmiert werden.

Die Parameter, die über die Maske des Subsystems eingestellt werden sollen (Masse des Pendels m, Seillänge l, Anfangswinkel φ_0 und Anfangswinkelgeschwindigkeit $\dot{\varphi}_0$), werden in den Blöcken des Subsystems selbst

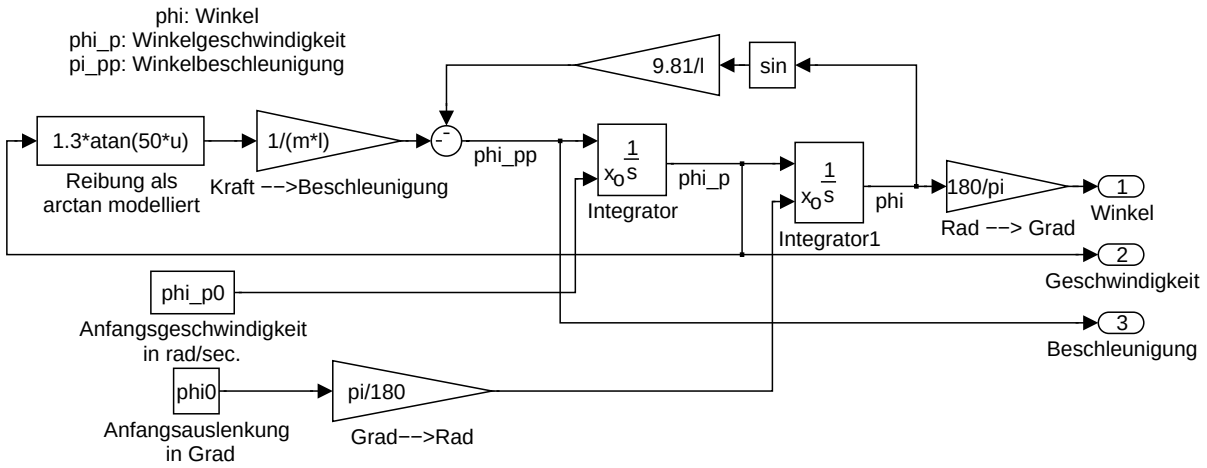


Abb. 8.2: *Maskiertes Subsystem Pendel, 'Gravitationspendel mit Reibung'*

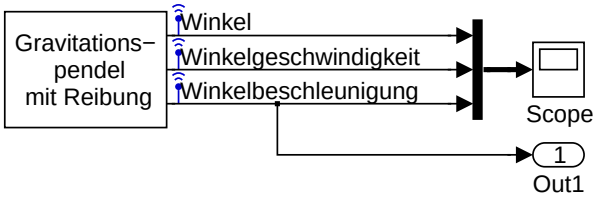


Abb. 8.3: *Simulink–Datei* `pendel.mdl`

nur als Variablen (m, l, $phi0$, phi_p0) eingetragen. Sie erscheinen dann wieder im *Initialization*-Register des Mask Editors unter *Variable*, wo sie als Platzhalter für die in den Textfeldern der Maske eingetragenen numerischen Werte dienen.

Abbildung 8.4 zeigt die Einträge im *Icon*- und *Initialization*–Register des Mask Editors.

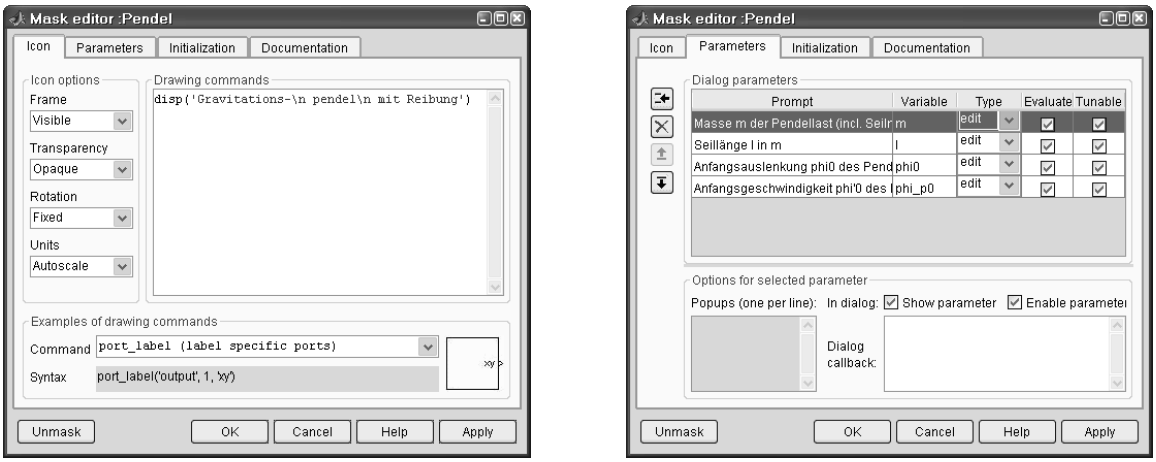


Abb. 8.4: *Icon- und Initialization–Register des Mask Editors des maskierten Subsystems Pendel*

Fein aufgelöste Kurvenverläufe ohne zu spürbare Rechenzeiterhöhung erhält man durch ein Heraufsetzen (z. B. auf 10) des *Refine factor* im Register *Data Import/Export* der *Configuration Parameters* Dialogbox.

Die drei Signale φ, $\dot{\varphi}$ und $\ddot{\varphi}$ können, wenn keine *Scope*- oder *To Workspace*-Blöcke verwendet werden sollen, mit Hilfe der Optionen *Save to workspace* des *Data Import/Export*-Registers der *Configuration Parameters* Dialogbox auf den Workspace geschrieben werden. Zur Speicherung der Zustandsvariablen φ und $\dot{\varphi}$ (Ausgänge der Integratoren) wird die Check Box *States* aktiviert. Die Zustandsvariablen werden dann in der Variable *xout* in dem gewählten Speicherformat abgelegt. Die Winkelbeschleunigung $\ddot{\varphi}$ wird zur Speicherung auf der obersten Modellebene auf einen *Outport*-Block geführt. Wird im *Data Import/Export*-Register die Check Box *Output* aktiviert, wird $\ddot{\varphi}$ in der Variable *yout* abgelegt. Mit dem Befehl

```
[size, x0, xstord] = pendel

size =
     2
     0
     1
     0
     0
     0
     2

x0 =
     0
    -1

xstord =
    'pendel/Pendel/Integrator1'
    'pendel/Pendel/Integrator'
```

kann überprüft werden, wie Simulink die Numerierung der Zustände vornimmt. Wurde als Speicherformat *Array* gewählt, können φ, $\dot{\varphi}$ und $\ddot{\varphi}$ mit den folgenden Befehlen in einer MATLAB-Figure (siehe Abb. 8.5) geplottet werden.

```
plot(tout,xout(:,1)*180/pi,'k',tout,xout(:,2),'k--', tout, yout,'k-.')
grid on
set(gca,'Ylim',[-65 65])
```

Alternativ können die Signale *Winkel*, *Winkelgeschwindigkeit* und *Winkelbeschleunigung* auch mittels *Signal Logging* auf den Workspace gespeichert werden. Dazu wird in den *Signal Properties* Dialogboxen der drei Signale die Check Box *Log signal data* aktiviert. Jedes der geloggten Signale wird dann mit einem -Icon markiert, wie in Abb. 8.3 zu sehen. Die Signale werden in der Variablen *logsout* gespeichert, die mittels

```
logsout.unpack('all')
```

entpackt wird. Anschließend können die Daten der Einzelsignale mit

```
plot(Winkel.Time,Winkel.Data*180/pi,'k',Winkel.Time,Winkelgeschwindigkeit.Data,'k--',Winkel.Time,
Winkelbeschleunigung.Data,'k-.')
```

geplottet werden.

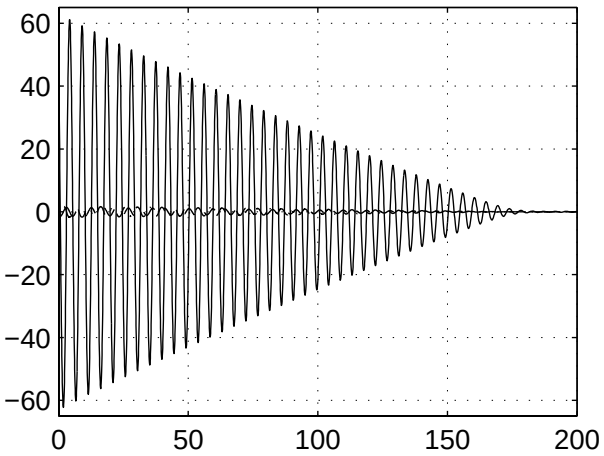


Abb. 8.5: *φ, $\dot{\varphi}$ und $\ddot{\varphi}$ dargestellt in einer* MATLAB*–Figure*

9 Buch-Kap. 9: Lineare und nichtlineare Systeme in Simulink

9.1 Modellierung einer Gleichstrom-Nebenschluss-Maschine (GNM) (Buch-Kap. 9.8.1)

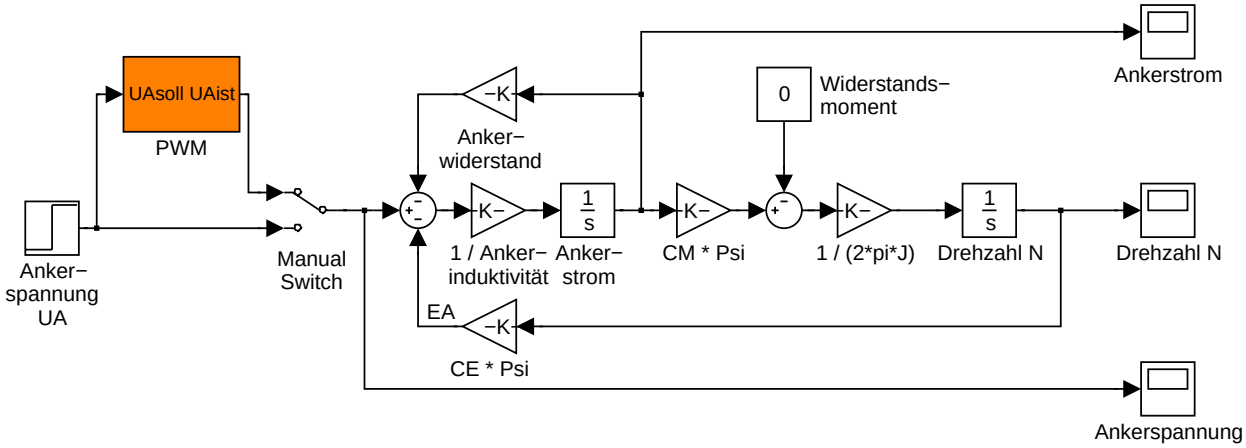


Abb. 9.1: *Modell des Ankerkreises der GNM:* `GNM_lsg.mdl`

Eine Initialisierung der Blöcke in `GNM_lsg.mdl` wird mit Hilfe des MATLAB Skripts `GNM_lsg_ini.m` vorgenommen. `GNM_lsg_ini.m` wird als Callback-Routine *Model initialization function* verknüpft.

```
% GNM_lsg_ini.m
% Kap. 9.8.1 und 9.8.2
% Initialisierungsdatei für GNM_lsg.mdl, PWM.mdl und GNM_lsg_bode.mdl

% Parameter der Gleichstromnebenschlußmaschine:

RA = 0.25; % Widerstand [Ohm]
LA = 0.004;     % Induktivitaet [H]
NA = 60; % Windungszahl =!= sqrt(LA/LE)*NE
psi = 0.04;          % Fluß [Vs]

J = 0.012; % Traegheitsmoment [kg m^2]
CM    = 2*NA/pi;
CE = (2*NA/pi)*2*pi; % Maschinenkonstante
Mw = 10;       % Widerstandsmoment [Nm]

Imax = 20;                  % Max. Ankerstrom [A]
Umax = 200;                 % Zwischenkreisspannung [V]

% Stellglied (PWM)

Fpwm = 5000; % Frequenz PWM [Hz]
TstrA = 1 / Fpwm / 2;     % Stromrichter als PT1
```

Mit der Datei `GNM_lsg_plot.m` werden die Spannung U_A, Strom i_A und Drehzahl N graphisch dargestellt, sie ist als Callback-Routine *Simulation stop function* verknüpft.

```
% GNM_lsg_plot.m
% Kap. 9.8.1 und 9.8.2
% Graphische Darstellung der Simulationsergebnisse für GNM_lsg.mdl

t = UA (:,1);
UAist = UA (:,2);
IAist = IA (:,2);
Nist = N (:,2) *60;

figure (1);
clf;

subplot (331);
if 1, % mit PWM
    stairs (t, UAist, 'm-');
    hold on;
plot ([0 0.05 0.05 max(t)], [0 0 1 1]*50, 'b--');
    axis ([0.048 0.052 -40 240]);
    legend ('Ist', 'Soll', 2);
else % ohne PWM
plot (t, UAist, 'm-');
    axis ([0 max(t) -40 240]);
end;
xlabel ('Zeit [s]');
title  ('Ankerspannung U_A [V]');

subplot (332);
plot (t, IAist, 'm-');
xlabel ('Zeit [s]');
title  ('Ankerstrom I_A [A]');

subplot (333);
plot (t, Nist, 'm-');
xlabel ('Zeit [s]');
title  ('Drehzahl N_ [U/min]');
```

9.2 Modellierung einer Pulsweitenmodulation (PWM) (Buch-Kap. 9.8.2)

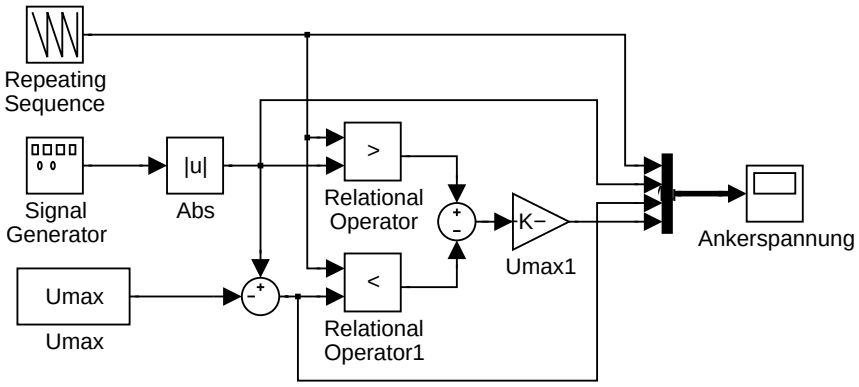


Abb. 9.2: *Dreipunkt-Pulsweitenmodulation (PWM):* `PWM.mdl` *bzw.* `GNM_lsg.mdl`*, Subsystem 'PWM'*

Der *Signal Generator* wird mit *sawtooth* (Sägezahn) betrieben; der Block *Abs* erzeugt daraus ein positives Dreieckssignal. Für die negative Dreiecksspannung wird dieses um $-U_{max}$ verschoben.[1)] Am Ausgang des *Sum*–Blocks steht ein Signal $\{-1,\ 0\ +1\}$ zur Verfügung, welches mit U_Z skaliert wird und so die Ausgangsspannung der PWM ergibt. Alternative Implementierungen sind möglich.

In der Lösungsdatei `GNM_lsg.mdl`, Abb. 9.1 ist die Pulsweitenmodulation bereits im Subsystem 'PWM' eingebracht. Über einen *Manual Switch* kann zwischen der herkömmlichen Step-Anregung und der PWM umgeschaltet werden.

[1)] Verschieben ist besser als eine Multiplikation des pos. Dreieckssignals mit -1, da letztere zu singulären Punkten (beide Dreiecksspannungen $= 0$) führt.

Die Datei `PWM_plot.m` stellt die Signale der PWM grafisch dar, sie ist als Callback-Routine *Simulation stop function* verknüpft.

```
% PWM_plot.m
% Kap. 9.8.2
% Graphische Darstellung der Simulationsergebnisse von PWM.mdl

t = UA (:,1) * 1000;
UAsoll = UA (:,2);
UAoben = UA (:,3);
UAunten = UA (:,4);
UApwm = UA (:,5);

figure (1);
clf;

subplot (311);
    plot (t, UAsoll, 'b--');
    hold on;
    stairs (t, UApwm, 'k-');
    plot (t, UAoben, 'm-');
    plot (t, UAunten, 'm-');
    axis ([0 max(t) -240 240]);
    legend ('Sollwert (Eingang)', 'Istwert (Ausgang)', 1);
xlabel ('Zeit [ms]');
title  ('PWM-Spannung [V]');
```

9.3 Aufnahme von Bodediagrammen (Buch-Kap. 9.8.3)

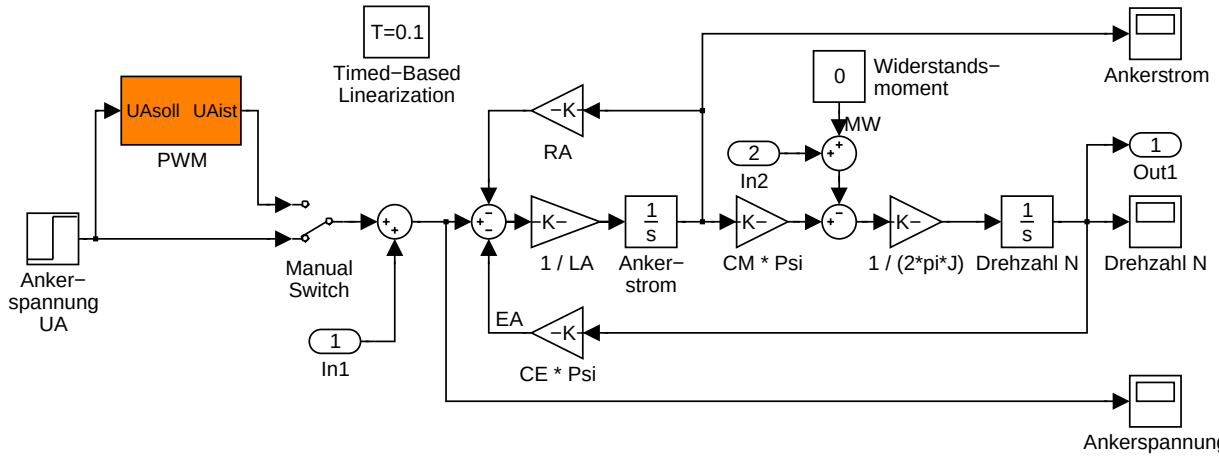


Abb. 9.3: *Ankerkreis der GNM mit PWM und* Inport*– und* Outport*–Blöcken zur Zustandsdarstellung mit* `linmod`*:* `GNM_lsg_bode.mdl`

Mithilfe des Blocks *Time-Based Linearization* wird `GNM_lsg_bode.mdl` zum angegebenen Zeitpunkt 0.1 s linearisiert und die Variable (Format *Structure*) *GNM_lsg_bode_Timed_Based_Linearization* im MATLAB-Workspace erzeugt. In der Datei `GNMbode.m` werden die Felder a bis d der Struktur ausgelesen; sie enthalten die Matrizen $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ und $\mathbf{D}$ der Zustandsdarstellung. Nach Umwandlung in zwei LTI-Objekte können die Einzel-Übertragungsfunktionen $N(s)/U_A(s)$ und $N(s)/M_W(s)$ in einem Bodediagramm dargestellt werden. `GNM_lsg_ini.m` und `GNMbode.m` sind als Callback-Routinen *Model initialization function* bzw. *Simulation stop function* verknüpft.

```
% GNMbode.m
% Kap. 9.8.3
% Erzeugung von Bodediagrammen von GNM_lsg_bode.mdl

% Aus der vom Block Time-Based Linearization geschriebenen
% Struktur GNM_lsg_bode_Timed_Based_Linearization werden die
% Matrizen der Zustandsdarstellung zwecks Namensverkürzung extrahiert
```

```
% u1 = UA, u2 = MW, y1 = N
A=GNM_lsg_bode_Timed_Based_Linearization.a;
B=GNM_lsg_bode_Timed_Based_Linearization.b;
C=GNM_lsg_bode_Timed_Based_Linearization.c;
D=GNM_lsg_bode_Timed_Based_Linearization.d;

% HINWEIS: A,B,C,D können natürlich auch mittels
% [A,B,C,D] = linmod('GNM_lsg_bode');
% erzeugt werden.
% Dieser Befehl darf jedoch nicht in GNMbode.m erfolgen, da
% GNMbode.m als Callback-Routine 'Simulation stop function' mit
% GNM_lsg_bode.mdl verknüpft ist (memory allocation error mit
% möglicher segmentation violation)!

% LTI-Objekt mit u1 und y1:
% (an sich kann das Bodediagramm direkt mit
% bode(A,B(:,1),C,D(:,1)) bzw.
% bode(A,B(:,2),C,D(:,2))
% erzeugt werden;
% sollen jedoch zusätzlich auch Frequenz-Grenzen, z.B. {1,10000}
% übergeben werden funktioniert bode nur mit einem LTI-Objekt statt
% Matrizen).
sysUA_N = ss(A,B(:,1),C,D(:,1));
% LTI-Objekt mit u2 und y1:
sysMW_N = ss(A,B(:,2),C,D(:,2));

figure(1)
bode(sysUA_N,{1,10000});
grid on
figure(2)
bode(sysMW_N,{1,10000});
grid on
```

10 Buch-Kap. 10: Abtastsysteme in Simulink

10.1 Zeitdiskreter Stromregler für die GNM (Buch-Kap. 10.5.1)

Die Parameter des zeitdiskreten Stromreglers ergeben sich zu

$$V_{RId} = \frac{T_1}{2\,T_\sigma V} = \frac{T_1}{2\,T/2V} = 5\,\Omega$$

$$T_{RId} = T_1 = 16\,ms$$

$$G_{RId}(s) = \frac{U_{Asoll}(s)}{I_{Asoll}(s) - I_A(s)} = V_{RId}\left(\frac{1}{1+sT_{RId}}\right) = 5\,\Omega \cdot \left(\frac{1}{1+s16\,ms}\right)$$

Die Transformation in den z-Bereich geschieht folgendermaßen

```
i_regler  = tf (V_RId*[T_RId 1], [T_RId 0]); % Erzeugt kontinuierliche Ü-Ftn.
i_diskret = c2d (i_regler, Ts, 'zoh');       % Umwandeln in Abtastsystem
[Bd, Ad]  = tfdata (i_diskret, 'v');         % Ausgabe Zähler-, Nennervektor
```

und ergibt die Regler-Übertragungsfunktion

$$G_{RId}(z) = \frac{U_{Asoll}(z)}{I_{Asoll}(z) - I_A(z)} = \frac{B_d(z)}{A_d(z)} = \frac{5z - 4.75}{z - 1}$$

Die Implementierung erfolgt mit einem *Discrete Transfer Fcn*-Block, siehe Abb. 10.1.

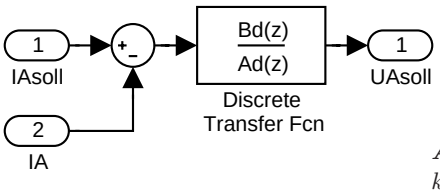


Abb. 10.1: *Zeitdiskreter Stromregler, Subsystem 'PI-Stromregler (BO) diskret' aus Lösungsdatei* `GNM_abtast.mdl`

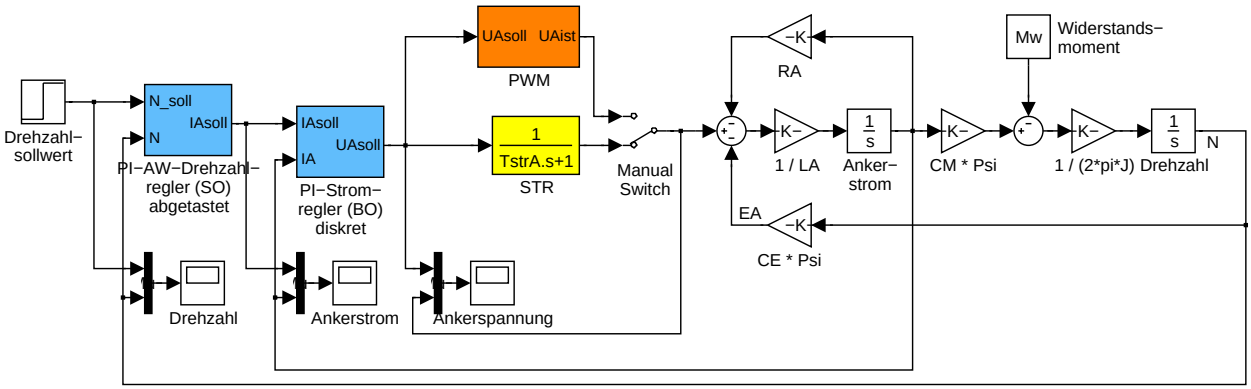


Abb. 10.2: *Lösungsdatei für Kap. 10.5.1 und 10.5.2:* `GNM_abtast.mdl`

Die Initialisierungsdatei `GNM_abtast_ini.m` wird als Callback-Routine *Model Initialization function* mit `GNM_abtast.mdl` verknüpft.

```
% GNM_abtast_ini.m
% Kap. 10.5.1 und 10.5.2
% Initialisierungsdatei für GNM_abtast.mdl

% Parameter der Gleichstrom-Nebenschluss-Maschine:
RA = 0.25; % Widerstand [Ohm]
LA = 0.004; % Induktivitaet [H]
NA = 60; % Windungszahl =!= sqrt(LA/LE)*NE
psi = 0.04;             % Fluß [Vs]

J = 0.012; % Traegheitsmoment [kg m^2]
CM    = 2*NA/pi;
CE = (2*NA/pi)*2*pi; % Maschinenkonstante
Mw = 10;       % Widerstandsmoment [Nm]

Imax = 20;                  % Max. Ankerstrom [A]
Umax = 200;                 % Zwischenkreisspannung [V]

% Stellglied (PWM)
Fpwm = 5000; % Frequenz PWM [Hz]
TstrA = 1 / Fpwm / 2;     % Stromrichter als PT1

% Stromregler
ViA = LA/(2*TstrA);       % Stromregler Verstaerkung
TiA = LA/RA; % Stromregler Zeitkonstante

% Drehzahlregler
N_soll = 1000 / 60;            % Solldrehzahl [rad/s]

VnA = 2*pi*J/(4*TstrA*CM*psi); % Drehzahlregler Verstaerkung
TnA = 8*TstrA;    % Drehzahlregler Zeitkonstante

% Stromregelung zeitdiskret
TiA = LA/RA;    % Zeitkonstante Ankerkreis

Tsi   = 0.0008;    % Abtastzeit [s]

ViAd     = LA/(2*Tsi/2);     % Stromregler Verstaerkung
i_regler  = tf (ViAd*[TiA 1], [TiA 0]); % Erzeugt kontinuierliche Ü-Ftn.
i_diskret = c2d (i_regler, Tsi, 'zoh'); % Umwandeln in Abtastsystem
[Bd, Ad]  = tfdata (i_diskret, 'v'); % Ausgabe Zähler-, Nennervektor

% Drehzahlregelung zeitdiskret
Tsn   = 0.0008;        % Abtastzeit [s]

VnAd   = 2*pi*J/(4*Tsn/2*CM*psi);    % Drehzahlregler Verstaerkung
TnAd   = 8*Tsn/2;     % Drehzahlregler Zeitkonstante
```

Mit der Datei `GNM_abtast_plot.m` werden die Spannung U_A, Strom i_A und Drehzahl N grafisch dargestellt. Sie wird als Callback-Routine *Simulation stop function* mit `GNM_abtast.mdl` verknüpft.

```
% GNM_abtast_plot.m
% Kap. 10.5.1 und 10.5.2
% Grafische Darstellung der Simulationsergebnisse von GNM_abtast.mdl

t = UA (:,1);
UAsoll = UA (:,2);
UAist = UA (:,3);
IAsoll = IA (:,2);
IAist = IA (:,3);
Nsoll = omega (:,2) *60;
Nist = omega (:,3) *60;

figure (1);
clf;

subplot (331);
plot (t, UAist, 'm-');
hold on;
    plot (t, UAsoll, 'b--');
    axis ([0 max(t) -40 240]);
```

```
    xlabel ('Zeit [s]');
title  ('Ankerspannung U_A [V]');
legend ('Ist', 'Soll', 4);

subplot (332);
plot (t, IAist, 'm-');
hold on;
plot (t, IAsoll, 'b--');
    axis ([0 max(t) -4 24]);
xlabel ('Zeit [s]');
title  ('Ankerstrom I_A [A]');
legend ('Ist', 'Soll', 3);

subplot (333);
plot (t, Nist, 'm-');
hold on;
plot (t, Nsoll, 'b--');
    axis ([0 max(t) -200 1200]);
xlabel ('Zeit [s]');
title  ('Drehzahl N_ [U/min]');
legend ('Ist', 'Soll', 4);
```

10.2 Zeitdiskreter Anti–Windup–Drehzahlregler für die GNM (Buch-Kap. 10.5.2)

Die Parameter des zeitdiskreten Drehzahlreglers ergeben sich zu

$$V_{RNd} = \frac{T_1}{2\,T_\sigma V} = \frac{T_1}{4\,(T/2)\,V} = \frac{2\pi\,J}{4\,(T/2)\,C_M\Psi} = 30.8425\,As$$

$$T_{RNd} = 4T_\sigma = 8\,T/2 = 3.2\,ms$$

$$G_{RNd}(s) = \frac{I_{Asoll}(s)}{N_{soll}(s) - N(s)} = V_{RNd}\left(\frac{1}{1 + s\,T_{RNd}}\right) = 30.8425\,As \cdot \left(\frac{1}{1 + s\,3.2\,ms}\right)$$

Die Implementierung mit zeitkontinuierlichen Übertragungsfunktionen (bzw. Integrator) bleibt erhalten. Sie wird durch je ein *Zero Order Hold*-Block am Eingang und Ausgang des Reglers mit der Abtastzeit $T_s = 800\,\mu s$ abgetastet, siehe Abb. 10.3.

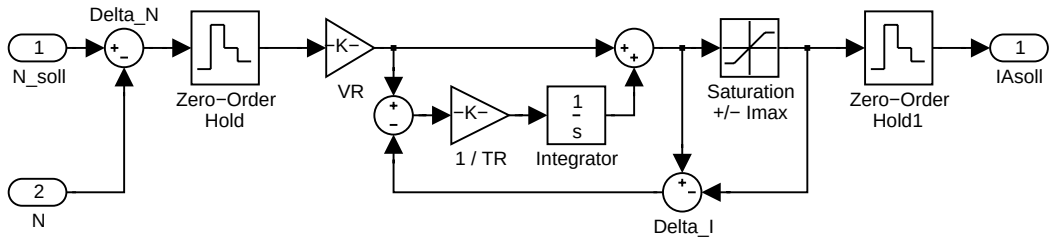


Abb. 10.3: *Zeitsynchron abgetasteter Drehzahlregler: Subsystem 'PI-AW-Drehzahlregler (SO) abgetastet' der Lösungsdatei* `GNM_abtast.mdl`

11 Buch-Kap. 11: Simulink-Regelkreise

11.1 Zustandsdarstellung GNM (Buch-Kap. 11.7.1)

Das Simulink-Modell `gnmregn.mdl` (Abb. 11.1) wird einfach um den *Mux*-Block und den *State Space*-Block erweitert, in den die vorher definierten Matrizen der Zustandsdarstellung eingegeben werden.

```
%% GNM in Zustandsdarstellung mit Widerstandsmoment
A = [ -RA/LA           -CE*PsiN/LA  ; ...
      CM*PsiN/(2*pi*J) 0            ; ...
    ];
B = [  1/LA    0          ; ...
       0      -1/(2*pi*J) ; ...
    ];
C = [  1  0 ; ...
       0  1   ...
    ];
D = zeros(size(C*B)) ;
```

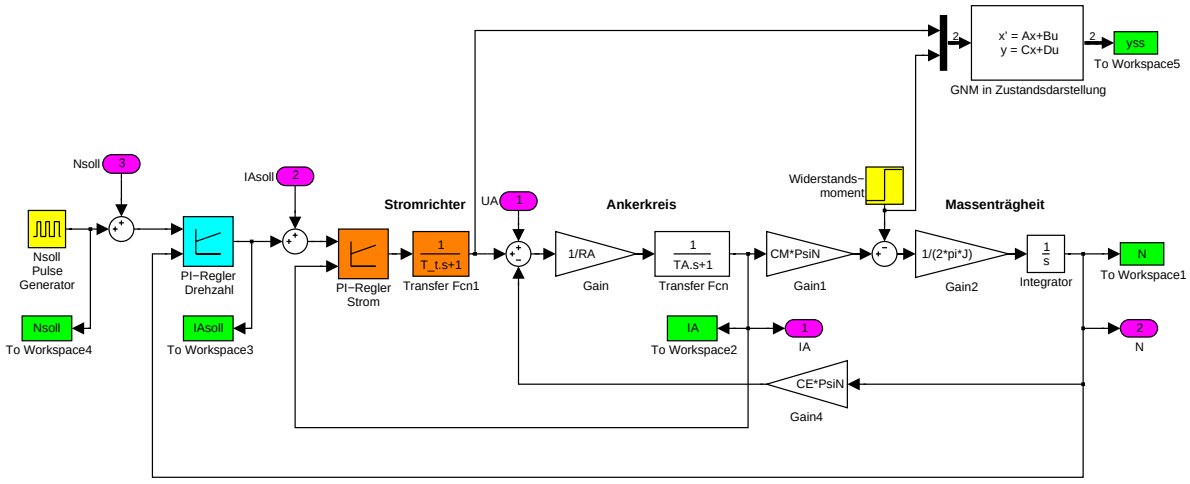


Abb. 11.1: *Simulink-Modell* `lsg_gnmregnss.mdl`

Einen Vergleich der beiden Darstellung zeigt z. B. Abb. 11.2:

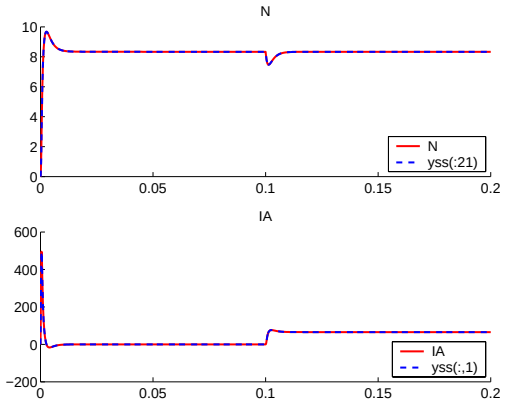


Abb. 11.2: *Sprungantwort und Störantwort von Drehzahl N und Ankerstrom IA bei Drehzahlregelung*

11.2 Systemanalyse (Buch-Kap. 11.7.2)

1. Die in Simulink erzeugte Sprungantwort sieht wie folgt aus:

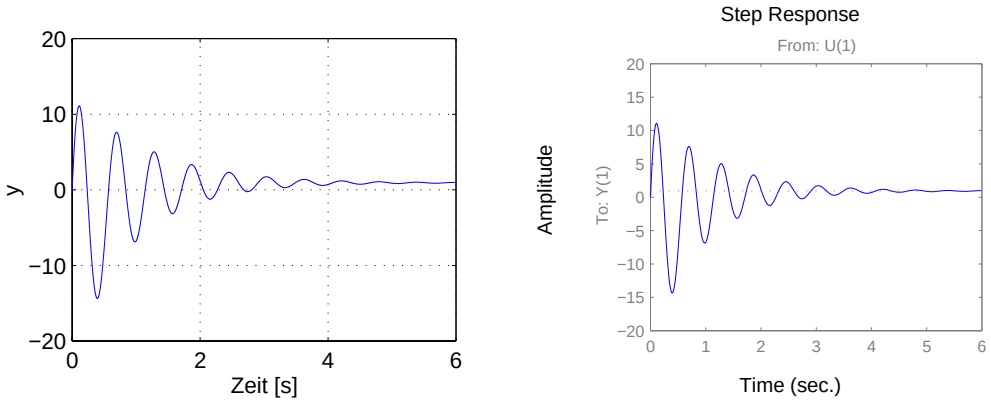


Abb. 11.3: *Sprungantwort in Simulink (links) und mittels step (rechts)*

2. Der nachfolgende Matlab–Code stellt eine Erweiterung der Initialisierungsdatei `system_ini.m` dar. Die mittels *step* erzeugt Sprungantwort ist bereits in Abb. 11.3 rechts dargestellt. Das Bodediagramm ist in Abb. 11.4 dargestellt.

```
%%  lsg_system_ini.m
%%  Kap 11.7.2
%%  Initialisierungsdatei zu system_bsp.mdl

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Es handelt sich hier nicht um ein technisches System, sondern um
% ein fiktives System, an dem der Regelungsentwurf in Matlab und die
% Simulation in Simulink eingeübt werden soll
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Tstop = 20;
max_step = 0.001;

% Regelstrecke
z = [1 1];                      % zeros
p = [-2 -3+10*i -3-10*i];       % poles
g = 2;                          % gain

% Rückführung des Systems (dies ist noch kein Regler !!!)
num = 2;
den = [0.02 1];

% Verstärkung
verst = 100;

% Lineare Systemanalyse mit linmod

[A,B,C,D] = linmod('system_bsp');    % Systemmatrizen extrahieren
sys_strecke = ss(A,B,C,D);           % ss-Objekt erstellen

% Simulation starten
sim('system_bsp')

% Sprungantwort mit Simulink und durch den Befehl step
figure
subplot(2,2,1)
plot(t,y); grid on;axis([0 6 -20 20]);
xlabel('Zeit [s]'); ylabel('y');
subplot(222)
step(sys_strecke);grid on;
% print -depsc sprung_system.eps

% Bodediagramm mit dem Befehl bode
```

```
figure
bode(sys_strecke)
% print -depsc bode_system.eps

% LTI-Viewer
ltiview(sys_strecke)
```

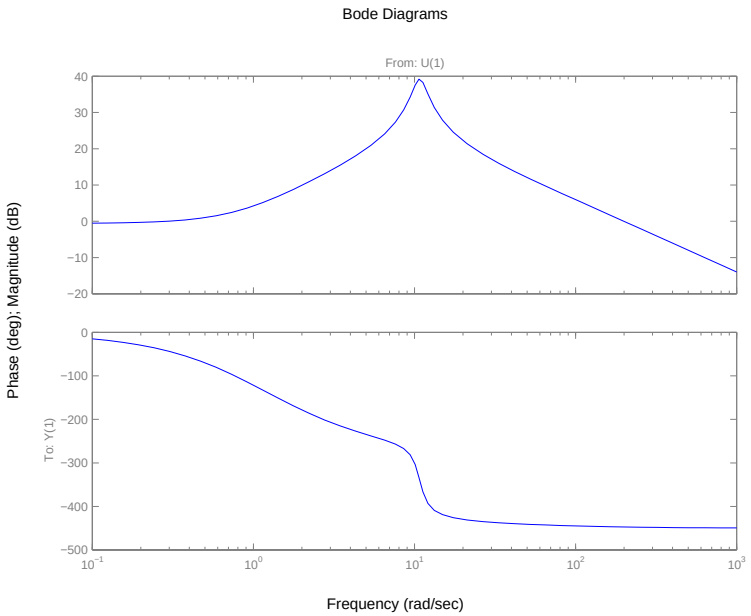


Abb. 11.4: *Bodediagramm des betrachteten Systems*

3. Das Ergebnis dieses Aufgabenpunktes ist das gleiche wie das der vorherigen Aufgabe, nur der Weg ist verschieden. Das Simulink–Modell muss durch Einfügen eines *Input Points* und eines *Output Points* (Kontextmenu - Linearization Points) wie in Abb. 11.5 erweitert werden. Anschließend kann der Simulink–LTI–Viewer gestartet und die geforderten Diagramme erzeugt werden.

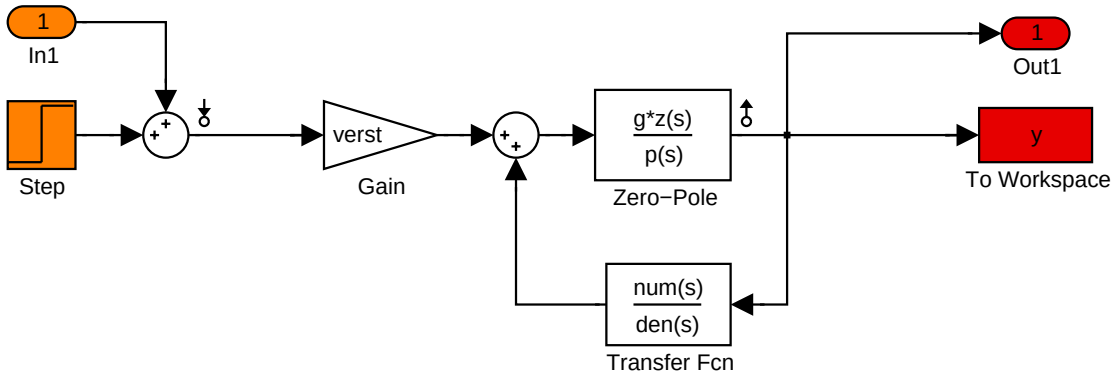


Abb. 11.5: *Für den Simulink–LTI–Viewer modifiziertes System*

11.3 Entwurf eines Kalman–Filters (Buch-Kap. 11.7.3)

1. Der Kalman–Filterentwurf ist aus nachfolgendem Matlab–Skript ersichtlich:

```
%%  lsg_sys_kalman_ini.m
%%  Kap 11.7.3
%%  Initialisierungsdatei zu lsg_sys_kalman.mdl

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Es handelt sich hier nicht um ein technisches System, sondern um
% ein fiktives System, an dem der Regelungsentwurf in Matlab und die
% Simulation in Simulink eingeübt werden soll
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Tstop = 10;
max_step = 0.01;

% Regelstrecke
z = [1 1];                      % zeros
p = [-2 -3+10*i -3-10*i];       % poles
g = 2;                          % gain

% Rückführung des Systems (dies ist noch kein Regler !!!)
num = 2;
den = [0.02 1];

% Verstärkung
verst = 100;

% Rauschen
Q = 1e-3;     % Varianz des Prozeßrauschens
R = 1e-3;     % Varianz des Meßrauschens

% Kalmanfilterentwurf

[A,B,C,D] = linmod('system_bsp');        % Systemmatrizen extrahieren
sys_strecke = ss(A,B,C,D);               % ss-Objekt erstellen
sys = ss(A,[B B],C,D);                   % ss-Objekt mit Rausch-Berücksichtigung
[kest, L, P] = kalman(sys,Q,R);          % Kalman-Filter entwerfen

% Simulation starten

sim('lsg_sys_kalman')

% Ergebnis graphisch darstellen

figure
subplot(2,2,1)
plot(t,y,t,ye,t,yw);axis([0 10 -15 15]);legend('gemessen','geschätzt','exakt')
xlabel('Zeit [s]');title('Schätzung der Ausgangsgröße')
subplot(2,2,2)
plot(t,xe);axis([0 10 -10 60]);
xlabel('Zeit [s]');title('Schätzung der Zustandsgrößen')
% print -depsc ausg_kalman.eps
```

Das Kalman–Filter kann wie in Abb. 11.6 in Simulink implentiert werden.

Die gesuchten Sprungantworten sind in Abb. 11.7 dargestellt.

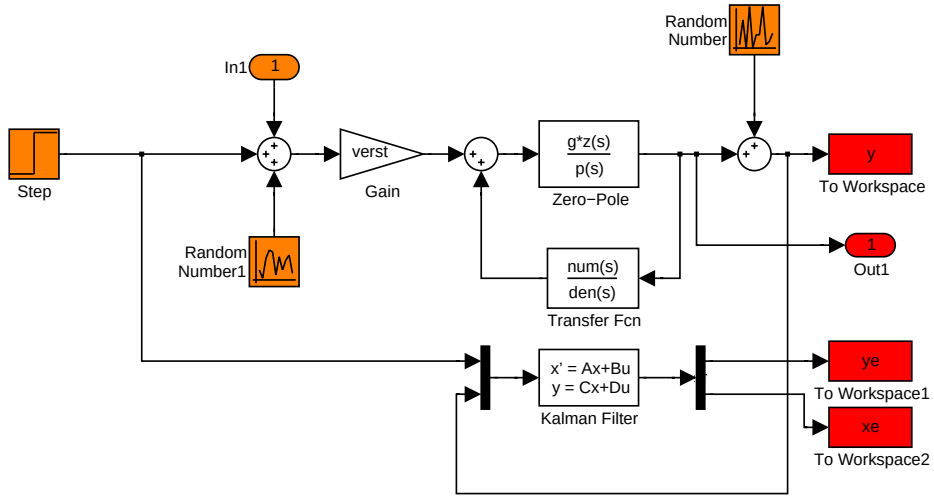


Abb. 11.6: *Kalman–Filter in Simulink*

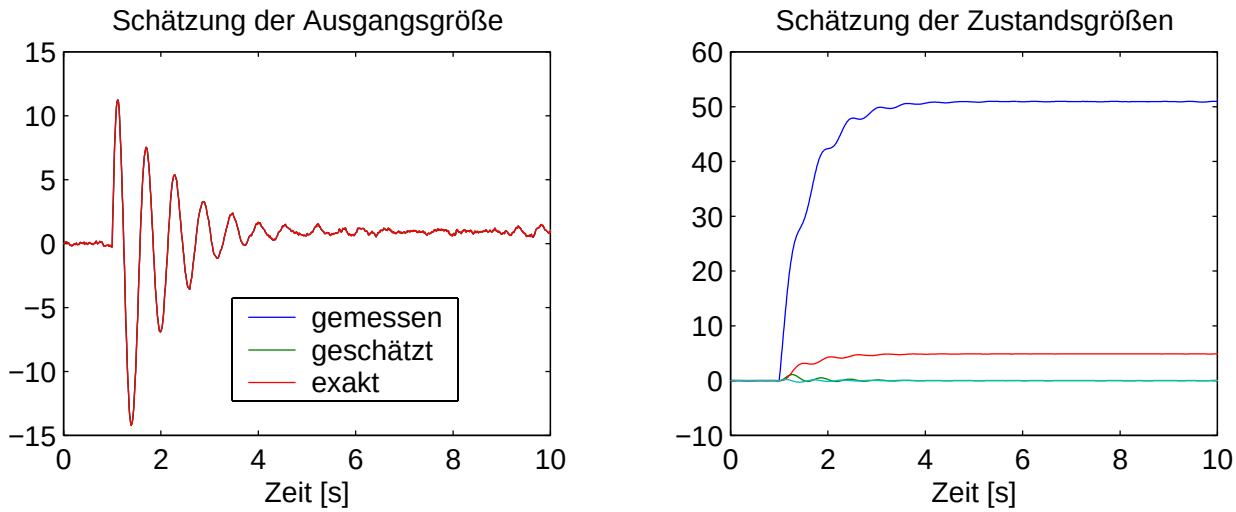


Abb. 11.7: *Geschätzte und gemessene Größen des Systems*

11.4 Entwurf eines LQ–optimierten Zustandsreglers (Buch-Kap. 11.7.4)

1. Der LQ–optimierte Reglerentwurf ist aus dem nachfolgenden Matlab–Skript ersichtlich:

```
%%  lsg_regelung_ini.m
%%  Kap 11.7.4
%%  Initialisierungsdatei zu lsg_regelung.mdl

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Es handelt sich hier nicht um ein technisches System, sondern um
% ein fiktives System, an dem der Regelungsentwurf in Matlab und die
% Simulation in Simulink eingeübt werden soll
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Tstop = 10;
max_step = 0.01;

% Regelstrecke
z = [1 1];                    % zeros
p = [-2 -3+10*i -3-10*i];     % poles
g = 2;                        % gain

% Rückführung des Systems (dies ist noch kein Regler !!!)
num = 2;
den = [0.02 1];

% Verstärkung
verst = 100;

% Rauschen
Q = 1e-3;     % Varianz des Prozeßrauschens
R = 1e-3;     % Varianz des Meßrauschens

% Kalmanfilterentwurf

[A,B,C,D] = linmod('system_bsp');        % Systemmatrizen extrahieren
sys_strecke = ss(A,B,C,D);               % ss-Objekt erstellen
sys = ss(A,[B B],C,D);                   % ss-Objekt mit Rausch-Berücksichtigung
[kest, L, P] = kalman(sys,Q,R);          % Kalman-Filter entwerfen

% Zustandsreglerentwurf durch LQ-Optimierung
% Entwurf durch den Befehl lqry

Q_lq = 1;                                % Gewichtsmatrix der Zustände
R_lq = 0.1;                              % Gewichtsmatrix der Stellgrößen
[k S e] = lqry(sys_strecke,Q_lq,R_lq);   % LQR-Reglerentwurf
Kv = -1/(C*inv(A-B*k)*B);                % Vorfaktor für stationäre Genauigkeit

% Simulation starten
sim('lsg_regelung')

% Ergebnis darstellen
figure; subplot(2,2,1)
plot(t,y);xlabel('Zeit [s]');title('gemessener Ausgang')
subplot(2,2,2)
plot(t,ye);xlabel('Zeit [s]');title('geschätzter Ausgang')
% print -depsc sprung_kalman.eps
```

2. Der Zustandsregler mit geschätzten Systemzuständen kann wie in Abb. 11.8 in Simulink implemetiert werden. Der Vorfaktor für stationäre Genauigkeit ist darin schon berücksichtigt.

3. Die geschätzte und gemessene Ausgangsgröße des geregelten Systems hat dann den in Abb. 11.9 dargestellten Verlauf.

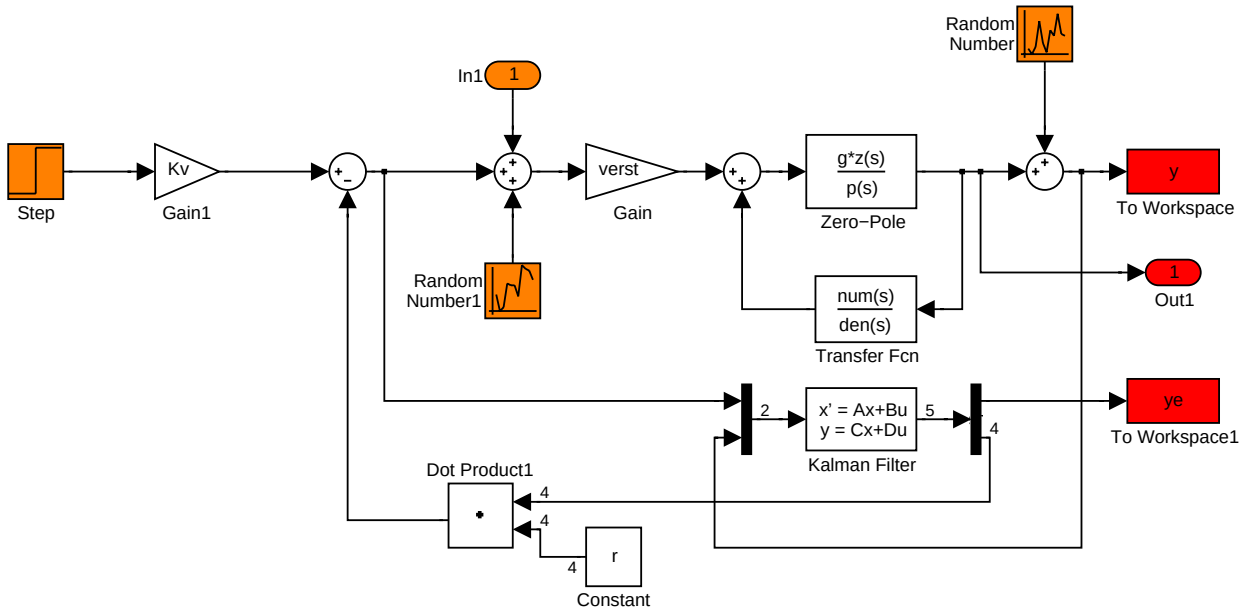


Abb. 11.8: Zustandsregelung mit Vorfaktor

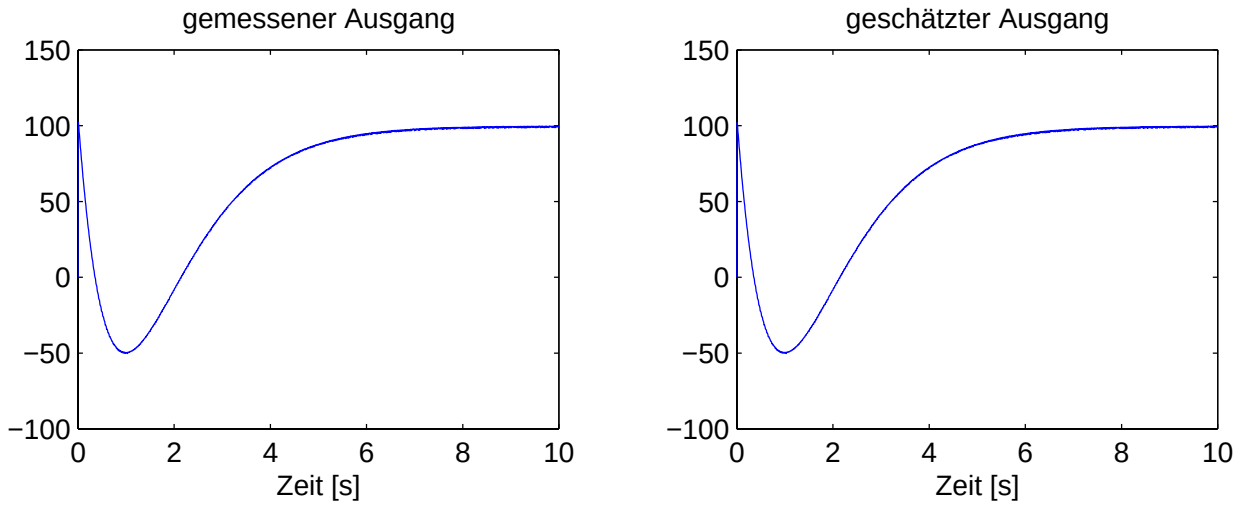


Abb. 11.9: Sprungantwort des geregelten Systems

12 Buch-Kap. 12: Stateflow

12.1 Mikrowellenherd (Buch Kap. 12.6.1)

Chart der Mikrowellensteuerung:

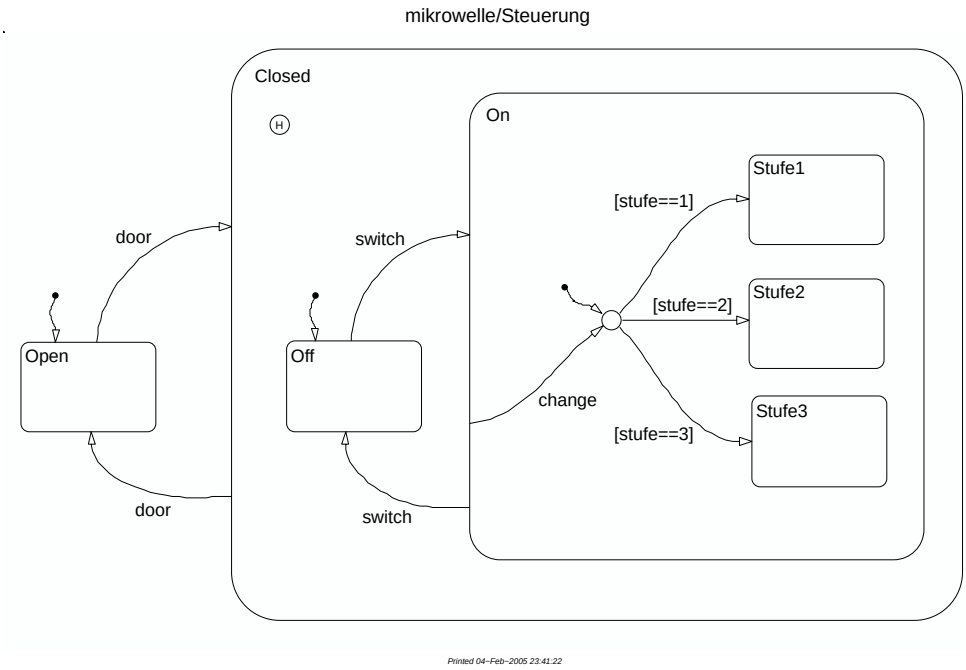


Abb. 12.1: *Chart der Mikrowellensteuerung;* `mikrowelle.mdl`

Zugehöriger Inhalt des Model Explorers:

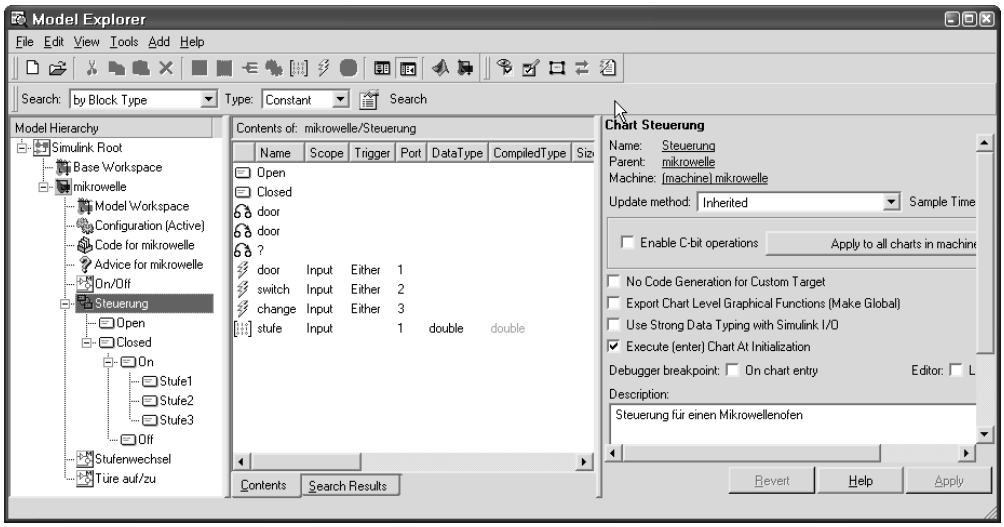


Abb. 12.2: *Inhalt des Model Explorers für die Mikrowellensteuerung*

Umgebendes Simulink Modell:

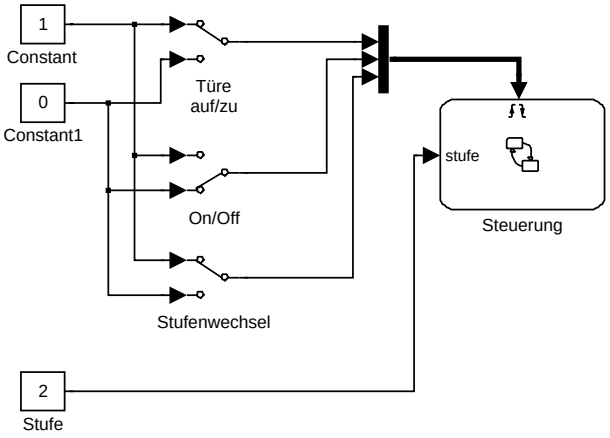


Abb. 12.3: *Simulink Modell der Mikrowellensteuerung*

Erklärung der Funktion:

Den Türstellungen *Offen* und *Geschlossen* entsprechen die Zustände `Open` und `Closed`. Nach der Initialisierung befindet sich die Steuerung im Zustand `Open`. Die Betätigung des Schalters `Türe auf/zu` löst das Event `door` aus und bewirkt einen Wechsel in den Zustand `Closed`. Nach der erstmaligen Aktivierung von `Closed` wird der *Substate* `Closed.Off` aktiv. Die Betätigung des Schalters `On/Off` löst das Event `switch` aus, wodurch eine Wechsel zwischen `Closed.Off` und `Closed.On` möglich ist. Bei der Aktivierung von `Closed.On` ist zunächst die *Default Transition* gültig und es wird der Zustand entsprechend der Bedingung `stufe==x` ausgewählt. Durch Betätigung des Schalters `Stufenwechsel` wird das Event `change` ausgelöst, wodurch die *Inner Transition* gültig wird. Durch vorherige Veränderung der Konstanten `Stufe` kann so während des Betriebs der Mikrowelle (Zustand `Closed.On`) ein Wechsel der Heizleistung erfolgen. Egal in welchem *Substate* von `Closed` sich die Steuerung befindet, durch Auslösen des Events `door` wird auch gleichzeitig der Zustand `Off` oder `On` verlassen. Die *History Junction* im *Superstate* `Closed` bewirkt, dass bei seiner erneuten Aktivierung der zuletzt aktive *Substate* aktiv wird; die *Default Transition* auf `Closed.Off` bleibt dann ohne Wirkung.

Im Simulink Modell wurde eine feste Integrationsschrittweite von 0.1 ms und ein Dauer von 100 s gewählt, so dass die Animation des *Charts* verfolgt werden kann.

Bitte beachten Sie, dass diese Lösung lediglich einen Vorschlag darstellt. Andere Lösungen können exakt die gleiche Funktionalität besitzen.

12.2 Zweipunkt-Regelung (Buch Kap. 12.6.2)

Chart des Zweipunkt-Reglers:

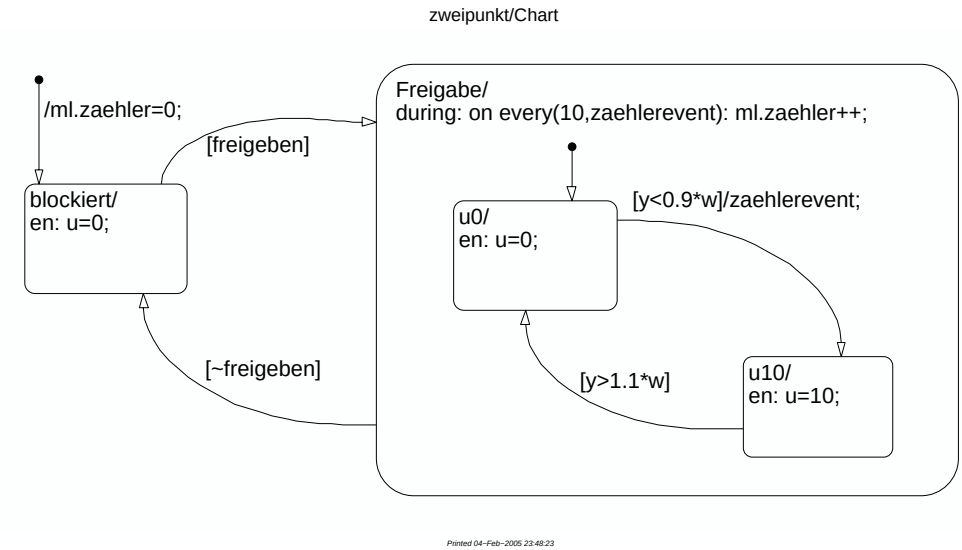


Abb. 12.4: *Chart der Zweipunkt-Regelung;* `zweipunkt.mdl`

Zugehöriger Inhalt des Model Explorers:

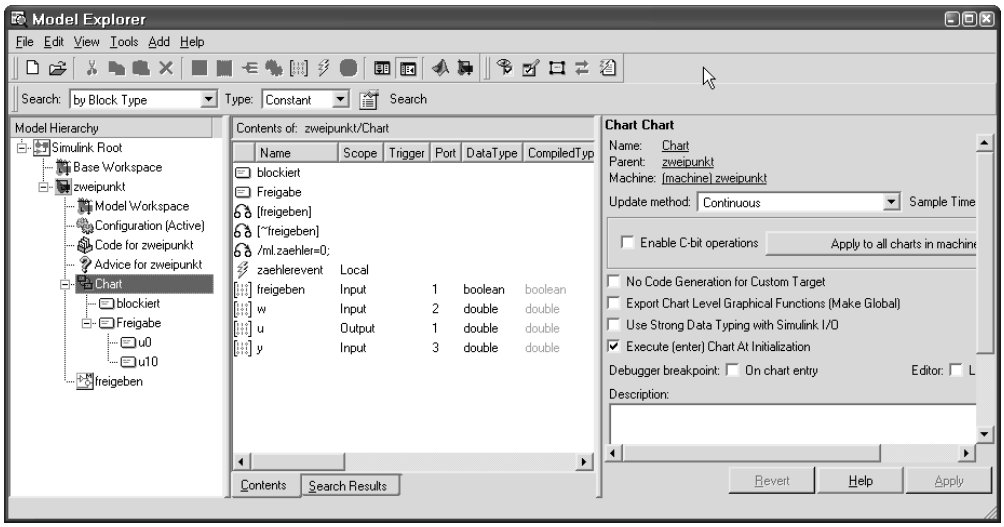


Abb. 12.5: *Inhalt des Model Explorers für die Zweipunkt-Regelung*

Umgebendes Simulink Modell:

Erklärung der Funktion:

Die Update Methode des *Charts* wurde im Menü *File/Chart Properties* auf *Continuous* gesetzt, da die Regelung bei jedem Integrationsschritt des Simulink-Modells ausgeführt werden muss. Die Zustände `blockiert` und `Freigabe` stellen den inaktiven bzw. aktiven Modus der Regelung dar. Nach der Initialisierung des *Charts* befindet sich die Regelung zunächst im Zustand `blockiert` und liefert die Stellgröße $u = 0$. Dies wird über die *Entry Action* des Zustands `blockiert` realisiert. Die *Default Transition* erzeugt die benötigte Zählervariable `zaehler` im MATLAB-Workspace.

Die Freigabe der Regelung erfolgt über die boolsche Variable `freigeben`, die in Simulink über manuelle Schalter von 0 auf 1 umgeschaltet werden kann. Wenn `freigeben` auf 1 wechselt, wird beim nächsten Integrationsschritt der *Superstate* `Freigabe` aktiviert. Er enthält die beiden *Substates* `u0` und `u10`, die den Stellgrößen $u = 0$ und $u = +10$ entsprechen. Die Stellgröße wird in den beiden *Entry Actions* auf $u = 0$

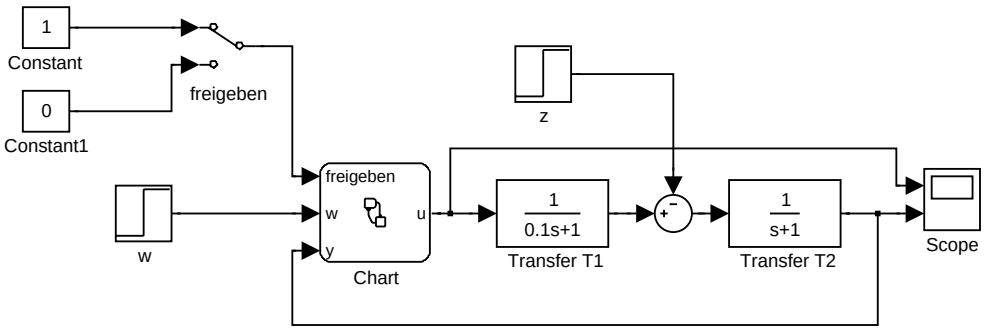


Abb. 12.6: *Simulink Modell der Zweipunkt-Regelung*

bzw. $u = +10$ gesetzt. Standardmäßig ist der *Substate* `u0` aktiv, was einer Stellgröße von $u = 0$ entspricht. Die Transition von `u0` nach `u10` ist dann gültig, wenn die Regelgröße y weniger als 90% des Sollwerts w beträgt. Als Transitionsaktion wird das lokale Event `zaehlerevent` ausgelöst, was in der *During Action* des *Superstates* `Freigabe` ausgewertet wird, da dieser Zustand während der Aktivität der Regelung immer auch aktiv ist. Diese *During Action* erhöht bei jedem 10ten Auftreten des Events `zaehlerevent` die Workspace-Variable `zaehler`. Die Transition von `u10` auf `u0` wird gültig, wenn die Regelgröße y größer als 110% des Sollwerts w ist. Die Stellgröße wird in der *Entry Action* von `u0` auf $u = 0$ gesetzt. Solange die Variable `freigeben` auf 1 bleibt, findet ein Wechsel zwischen den Zuständen `u0` und `u10` mit den Stellgrößen $u = 0$ und $u = +10$ gemäß einer Zweipunkt-Regelung statt. Am Ende der Simulation enthält die Workspace-Variable `zaehler` ein Zehntel aller durchgeführten Wechsel zwischen `u0` und `u10`.

Im Simulink-Modell kann mit dem manuellen Schalter während der Simulation die Regelung aktiviert und deaktiviert werden. In den *Step*-Blöcken `w` und `z` können Sollwert und Störgröße eingestellt werden. Im *Scope* können Regel- und Stellgrößenverlauf betrachtet werden.