Einführung in moderne Simulationstechniken
Kursteil: Matlab und Simulink

Felix Goßmann

Institut für Steuer- und Regelungstechnik
Fakultät für Luft- und Raumfahrttechnik
Universität der Bundeswehr München

Kontakt

Dozent:

Felix Goßmann, M.Sc.

Institut für Steuer- und Regelungstechnik

http://www.unibw.de/lrt15

Gebäude 41/300, Raum 2311

Terminvereinbarung per E-Mail:

felix.gossmann@unibw.de

Kursteil: Matlab und Simulink

- **Inhalt:**
  - Einführung in Matlab
  - Einführung in Simulink
  - Übungsaufgaben
- **Literatur:**
  - Angermann, Beuschel, Rau, Wolfrath, *Matlab – Simulink – Stateflow*,Oldenburg Verlag, München, Oldenburg Verlag, 6. Auflage 2009, mit CD-ROM. http://www.matlabbuch.de
  - Stein, *Einstieg in das Programmieren mit MATLAB*, Hanser Verlag,3. Auflage 2011. http://www.hanser.de
  - u.v.a.m. In der Uni-Bibliothek oder im Buchhandel
- **Online-Ressourcen:**
  - www.matlabcentral.com
  - www.mathworks.com

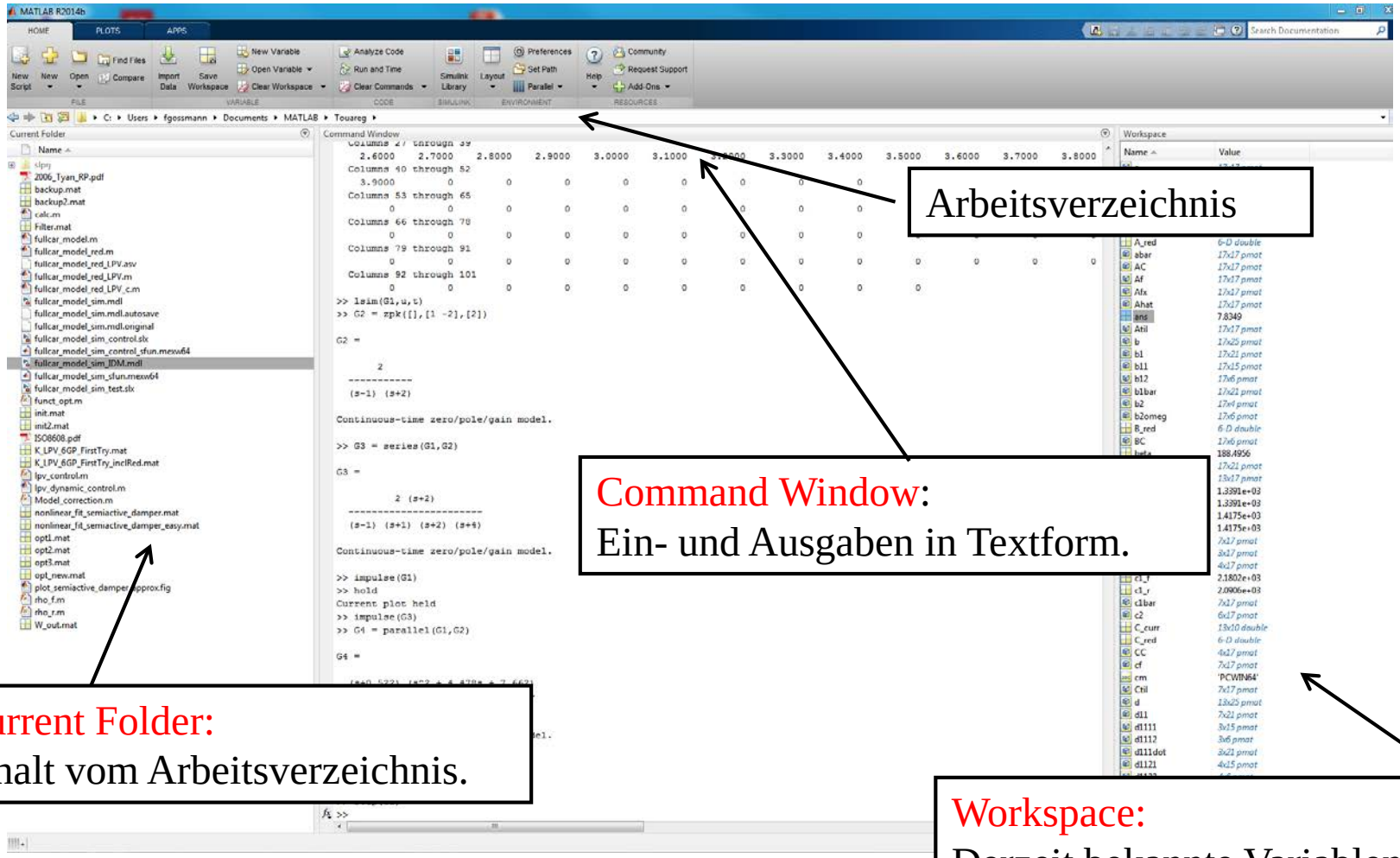
Einführung in Matlab - Inhalt

- Was ist Matlab ?
- Standardansicht
- Matlab als Taschenrechner
- Datenein- und -ausgabe
- Programmierung
- Anwendung in der Regelungstechnik

Was ist Matlab ?

- Programmiersprache und -umgebung für wissenschaftlich–technische Berechnungen; Hersteller: *The MathWorks, Inc.*, Natick, MA, USA (gegr. 1984)
- Entstanden in den 1970er Jahren, auf Grundlage von Fortran77- Softwarepaketen aus dem Bereich der Numerischen Linearen Algebra.
- Basisdatentyp: Matrix -> „**Mat**rix **lab**oratory"
- Matlab = Basismodul + Toolboxes
  - Basismodul:
  - Kontroll- und Datenstrukturen (an C angelehnt),
  - E/A, einschl. Schnittstelle zu externer Software (C, Fortran, Java) und Hardware (seriell),
  - Math. Funktionen,
  - 2D-, 3D-Graphik,...
  - Toolboxes:
  - Simulink,
  - Control System Toolbox,
  - Optimization Toolbox,...

Standardansicht



Matlab als Taschenrechner: Allererste Schritte

- Rechenoperationen:

`+`	Addition	`-`	Subtraktion
`*`	Multiplikation	`/`	Division
`^`	Potenzieren	`'`	Transposition

- Konstanten:

pi	Kreiszahl
exp	E-Funktion (Euler‘sche Zahl)
inf	Unendlich

……

- Variablen:

Zuweisung an beliebige Variablen möglich:

```
>> var_1 = 5
var_1 = 5
```

Standardergebnisvariable *ans*

Werden standardmäßig im globalen Workspace definiert

Matlab als Taschenrechner: Allererste Schritte

```
>> 2+3^2 % + 3      alles nach '%' is Kommentar
ans = 11          % Ergebnis in Variable 'ans' -> WS


>> ans * ans;     % Ergebnis in 'ans'. ';': keine Ausgabe


>> x1 = sin(pi/2)% Ergebnis in 'x1'. 'ans' nicht modifiziert
x1 = 1


>> X1 = x1 + j    % i, j: imaginäre Einheit
X1 = 1.0000 + 1.0000i
```

Matlab als Taschenrechner: Allererste Schritte

```
>> x1
x1 = 1              % Bezeichner case-sensitive

>> 1x = 5           % Bezeichner beginnen mit Buchstaben
??? 1x = 5
     |
Error: Unexpected MATLAB expression.

>> clear X1         % Variable 'X1' löschen
>> clear all        % alle Variablen löschen
>> clc              % Command Window leeren
```

Matlab als Taschenrechner: Vektoren

```
>> v = [1 2 3];  % Zeilenvektor. Leerzeichen & ',' trennen
>> [1;2;3]       % Spaltenvektor, auch mit Transpostion v‘
ans =  1         % berechenbar, ';‘ beendet Zeile
       2
       3
>> [1:3]         % Intervall ganzer Zahlen
ans = 1     2     3

>> [1:0.5:3]     % Angabe der Schrittweite
ans = 1.0000    1.5000    2.0000    2.5000    3.0000
>> ans(2:3)      % Zugriff auf Komponenten eines Vektors
ans = 1.5000    2.0000

>> ans(1)
ans = 1.0000
```

Matlab als Taschenrechner: Vektoren

Anwendung Arithmetischen Rechenoperationen (sofern von Dimension möglich):

```
>> [1;2;3] + [1 2 3]      % Dimensionen passen nicht
Error using  +
Matrix dimensions must agree.

>> [1;2;3]‘ - [1 2 3]     % Dimensionen mittels Transposition
ans = 0     0     0       % angepasst

>> [1;2;3] * [1 2 3]     % Multiplikation genau umgedreht
ans = 14

>> [1 2 3]'*[1; 2; 3]
Error using  *
Inner matrix dimensions must agree.
```

Matlab als Taschenrechner: Vektoren

Aufgabe:

Erzeuge die Vektoren $a = \begin{pmatrix} 8 \\ 1 \\ 7 \\ 4 \end{pmatrix}$ und $b = \begin{pmatrix} 6 & 5 & 8 & 2 \end{pmatrix}$

- Wende die folgenden Befehle auf die beiden Vektoren an und versuche deren Funktion herauszufinden:
  size(), length(), max(), min(), mean(), sum()

- Lasse dir jeweils das zweite, dritte und beide Elemente von *a* und *b* ausgeben
- Welche der folgenden Rechenoperationen lassen sich ausführen:
  *a*b, a'*b, a+b, a'+b*

- Was berechnen die Funktionsaufrufe max(a,b') bzw. min(a,b')?

Matlab als Taschenrechner: Vektoren

Lösung:

```
>> size(a)        % Spaltenvektor ist 1 x n - Matrix
ans = 4     1     % Gibt Dimension des Vektors an

>> size(b)        % Zeilenvektor analog
ans = 1     4

>> length(a)      % Anzahl der Komponenten eines Vektors
ans = 4

>> max(a)/ min(b) % Größter bzw. kleinster Eintrag im Vektor
ans = 8 / 1

>> mean(a) / sum(a) % Durchschnittswert und Summe aller Vektor-
ans = 5 / 20        % Einträge
```

Matlab als Taschenrechner: Vektoren

Lösung:

```
>> a(2)                    % Analog für b anwendbar
ans = 1

>> a(3)
ans = 7

>> a(2:3)                  % Bei Abfrage mehrere Elemente bleibt
ans = 1                    % Art des Vektors (Zeile o. Spalte)
      7                    % erhalten
```

- a‘*b und a+b lassen sich nicht ausführen, da Dimensionen nicht passend, a*b und a‘+b hingegen schon
- Bei der Eingabe von zwei oder mehreren Vektoren geben die Funktionen den größten bzw. kleinsten Vektor aus

Matlab als Taschenrechner: Matrizen

```
>> m = [1,0,0;1:3;5:2:9] % 3 x 3 – Matrix, Zeilen- und
m =  1     0     0       % Spaltentrennung analog zu Vektoren
     1     2     3
     5     7     9

>> m*[1;2;3]     % Multiplikation Matrix mit Spaltenvektor
ans = 1
      14
      46
>> m(2,3)        % Zugriff auf Komponenten einer Matrix
ans = 3          % 1. Angabe Zeile, 2. Angabe Spalte
>> m(3,2)
ans = 7
```

- Arithmetische Rechenoperationen zwischen Matrizen ebenfalls möglich, Anwendung analog zu Vektoren (sofern Dimensionen stimmig)

Matlab als Taschenrechner: Matrizen

Aufgabe:

Erzeuge die Matrizen $A = \begin{pmatrix} 4 & 3 & 2 \\ 5 & 3 & 1 \\ 2 & 4 & 1 \end{pmatrix}$ und $B = \begin{pmatrix} 1 & 1 & 5 \\ 3 & 2 & 4 \\ 2 & 1 & 3 \end{pmatrix}$

- Welche der in der vorherigen Aufgabe benutzten Befehle lassen sich auch auf Matrizen anwenden. Wenn ja, versuche herauszufinden was sie berechnen

  size(), length(), max(), min(), mean(), sum()
- Lasse dir die Elemente auf der Hauptdiagonalen von A & B ausgeben. Greife auf alle Elemente der 1. Spalte einzeln zu
- Wende zusätzlich die folgenden Befehle auf beide Matrizen an und versuche deren Funktionsweise zu ermitteln:

  diag(), trace(), eig(), inv()
- Berechne $A + B$, $A \cdot B$, $A.\cdot B$, A/B und $A./B$, ermittele den Unterschied zwischen $A \cdot B$ und $A.\cdot B$ (bzw. A/B und $A./B$)
- Versuche die Funktion von *zeros(n,n)*, *ones(n,n)* und *eye(n)* herauszufinden

Matlab als Taschenrechner: Matrizen

Lösung:

```
>> size(A)          % Gibt Dimension der Matrix an
ans = 3     3
>> length(A)        % Gibt größte „Ausdehnung“ der Matrix an
ans = 3

>> max(A)/ min(A) % Größter bzw. kleinster Eintrag aller
ans = 5 4 2         % Spaltenvektoren
/
ans = 2 3 1

>> mean(a) / sum(a)       %Durchschnittswert bzw. Summe aller
ans = 3.667 3.333 1.33   % Einträge der Spaltenvektoren
/
ans = 11    10    4
```

Matlab als Taschenrechner: Matrizen

Lösung:

```
>> A(1,1) / >> A(2,2)    % Hauptdiagonale
ans = 4   / ans = 3
>> A(1,2)                % zweite Spalte in der ersten Zeile
ans = 3                  % usw
>> diag(A)               % Gibt Elemente der Hauptdiagonale
ans = 4                  % als Spaltenvektor aus
      3
      1
>> trace(A)              % Spur der Matrix (Summe der
ans = 8                  % Elemente auf Hauptdiagonale)
>> eig(A)                % Eigenwerte der Matrix
ans =  8.6617 + 0.0000i
      -0.3309 + 1.2737i
      -0.3309 - 1.2737i
>> inv(A)                % Inverse der Matrix (sofern möglich)
```

Matlab als Taschenrechner: Matrizen

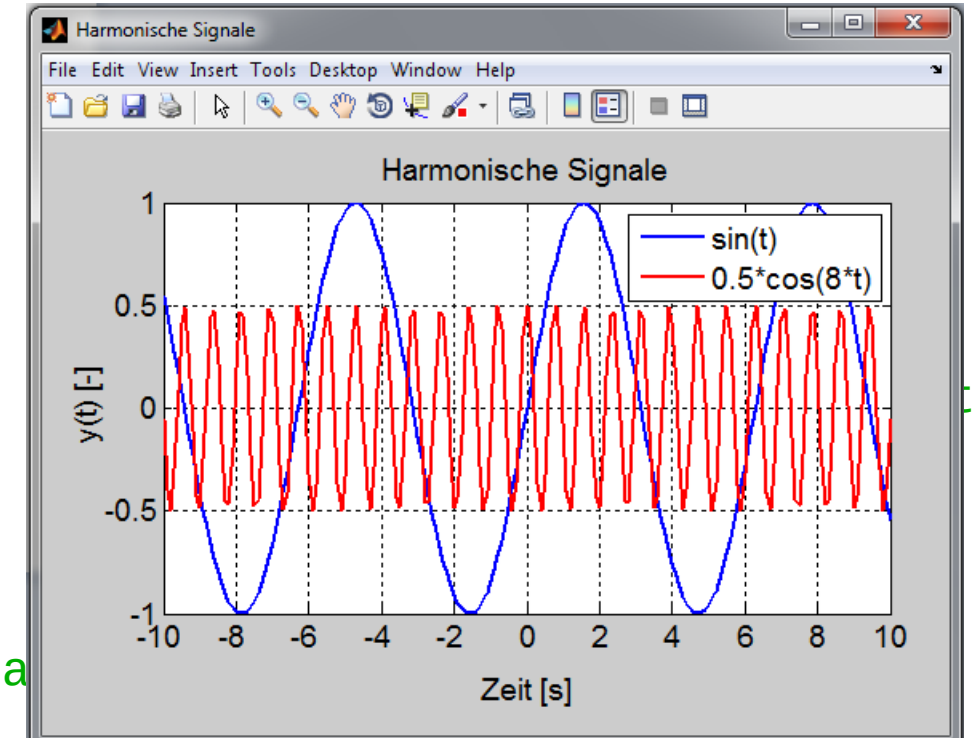
Lösung:

```
>> A*B                                  % Normale Matrizenmultiplikation
ans = 17     12     38
      16     12     40
      16     11     29
>> A.*B                                 % Multiplikation der einzelnen
ans = 4      3     10                  % Elemente
      15     6      4
      4      4      3
>> A/B                                          % Entspricht A*inv(B)
ans = -0.7500    2.7500   -1.7500    % Lässt sich analog zu oben
      -1.5000    2.5000   -0.5000    % auch Elementweise durch-
       0.2500    5.7500   -7.7500    % führen
```

- *zeros()* und *ones()* erzeugt Matrizen in gewünschter Dimension, die nur aus Nullen oder Einsen bestehen
- *eye()* erzeugt die Einheitsmatrix in gewünschter Dimension

Matlab als Taschenrechner: 2D-Graphiken

```
>> figure(1)
>> t = [-pi:2*pi/100:pi];
>> x = sin(t);
>> y = 0.5 * cos(8*t);
>> plot(t,x,t,y);
>> plot(t,x);
>> hold on
>> plot(t,y,'r');
>> grid on
>> % Weiteres: legend, xla
```




```
>> close(1)                    % Graphik-Fenster 1 schließen
>> close all                   % alle;
```

So viele Sachen: Hilfe!

```
>> help sin    % einfache Hilfe zur Funktion 'sin'
SIN    Sine of argument in radians.
    SIN(X) is the sine of the elements of X.

    See also asin, sind.

    Overloaded methods:
       codistributed/sin

    Reference page in Help browser
       doc sin  % ausführlichere Hilfe

>> helpdesk     % Auswahl an Hilfefunktionen, Demos, Suche
                % nach Funktionennamen, usw.
```

Daten Ein- und Ausgabe

```
>> mkdir meintest    % Anlegen eines Verzeichnisses
>> cd meintest/      % als Arbeitsverzeichnis wählen
>> x=1;y=1;z=1;
>> save xydatei x y % x, y -> Datei xydatei.mat im Arbeitsv.
>> save xyzdatei     % x, y, z -> Datei xyzdatei.mat
>> x=2;y=2;z=2;

>> load xydatei
>> [x y z]
ans = 1     1     2
>> load xyzdatei
>> [x y z]
ans = 1     1     1

>> cd ..             % Arbeitsverzeichnis wechseln
>> rmdir meintest/   % Testdaten entfernen
```

Programmierung

- Programme (“Skripte”) und Funktionsdefinitionen in Dateien vom Typ *.m
- Nutzung im Command Window über Rumpf des Dateinamens
- **Programme:**
  - Abarbeitung so, als würde Dateiinhalt interaktiv eingegeben
  - Neben Rechenoperationen auch Verwendung von anderen Progammierungselementen wie Schleifen (*for, while, if*) möglich
- **Funktionsdefinitionen:**

```
function wert = fname(arg1,arg2,...)
% H-Zeile (Hilfetext)
<Matlab-Anweisungen, die auf Argumente angewandt
werden>
wert = ... % Zuweisung des Funktionswertes
end        % Ende der Definition von fname
```

  - Namen der Argumente (arg1,...) und des Funktionswertes (wert): frei; wert ist nur lokal sichtbar (nicht im Workspace)
  - function, end: Schlüsselworte
  - Funktionswert: auch Vektor/Matrix möglich
  - H-Zeile: wird im Command Window mit durch >> help ausgegeben

Verwendung von Schleifen:

For-Schleifen:

```
for i = 1:1:10          % Für i von 1 bis 10 mit der
   „Operationen“        % Schrittweite 1, bei keiner Angabe
end                     % der Schrittweite wird autom. 1
                        % verwendet
```

If-Schleifen:

```
if i = n                % Wenn i = n, dann …
   „Operationen“
elseif i = m            % Wenn i = m, dann …
   „Operationen“
else                    % In allen anderen Fällen …
   „Operationen“
end
```

Verwendung von Schleifen:

While-Schleifen

```
while i = j                    % Solange i = j, dann …
   „Operationen“
end
```

- Der Schleifenaufruf und –ende benötigen kein Semikolon, alle anderen Operation innerhalb schon!

Beispiel eines Programms:

- Editor öffnen (z.B. Matlab-Editor: >> edit)
- Datei meinp.m mit folgendem Inhalt erzeugen:

```
figure(1)
hold on                          % man teste auch 'off'
t = (1:1000)/100;
for k = 1:3
plot(t, k*sin(k*t));
pause(1);
end
clear all
close all
```

- Programm ausführen:

```
>> meinp                         % Ohne Semikolon („;“)!
```

Beispiel einer Funktion:

- Editor öffnen (z.B. Matlab-Editor: >> edit)
- Datei meinfun.m mit folgendem inhalt erzeugen:

```
function wert = meinfun(x)
% Ausgabe: [arith. Mittel, Minimum, Maximum]
a = mean(x);
wert = [a min(x) max(x)];
end
```

- Funktion im Command Window verwenden:

```
>> help meinfun
Ausgabe: [arith. Mittel, Minimum, Maximum]

>> meinfun([1:3])
ans = 2     1     3
```

Aufgaben:

- Schreibe ein Skript, dass die Funktion $f(x) = x^2 + 4 + e^x$ auf dem Intervall $x = [-5{:}\,5]$ mit einer Schrittweite von $\mathrm{n} = 0.1$ berechnet und anschließend graphisch ausgibt (Hinweis: e^x wird mit *exp(x)* berechnet). Experimentiere anschließend mit der Schrittweite und beobachte das Ergebnis

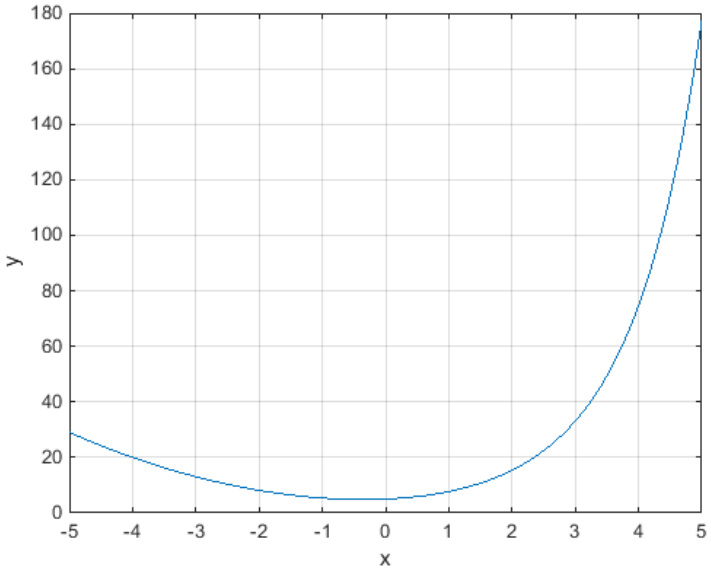
- Schreibe eine Funktion, die den größten Wert auf der Hauptdiagonale einer belieben symmetrischen Matrix ausgibt

- Schreibe eine Funktion, die die Summe der Beträge *abs()* aller Vektorelemente berechnet (Verwende eine *for*-Schleife)

Lösung:

- Schreibe ein Skript, dass die Funktion $f(x) = x^2 + 4 + e^x$ auf dem Intervall $x = [-5{:}\,5]$ mit einer Schrittweite von $\mathrm{n} = 0.1$ berechnet und anschließend graphisch ausgibt.

```
x = [-5:0.1:5];
y = x.^2+4+exp(x));
plot(x,y);
xlabel('x');
ylabel('y');
grid on;
```



Bei Senkung der Schrittweite wird der Plot ungenauer/grober

Lösung:

- Schreibe eine Funktion, die den größten Wert auf der Hauptdiagonale einer belieben symmetrischen Matrix ausgibt

```
function m = maxWertHauptdiag(X)

% X ist eine Matrix. Es wird der maximale Wert auf deren
% Hauptdiagonale berechnet

d = diag(X);
m = max(d);
end
```

Aufruf:

```
>>maxWertHauptdiag(X)                % X aus vorheriger Aufgabe
ans = 4
```

Lösung:

- Schreibe eine Funktion, die die Summe der Beträge *abs()* aller Vektorelemente berechnet (Verwende eine *for*-Schleife)

```
function summe = AbsVektorSumme(v)

% v ist ein Vektor. Es soll die Summe der Absolutwerte im
% Vektor berechnet werden

summe =  0;                    % Vordefinition der Summe

for i = 1:length(v)            % Schrittweite nicht definiert,
summe = summe+abs(v(i));       % daher 1
end                            % aufsummieren der einzelnen
                               % Elemente
end
```

Anwendung in der Regelungstechnik

- Übertragungsfunktion eines SISO – Systems:

Zählerpolynom

$$G(s) = \frac{Z(s)}{N(s)} = \frac{z_m s^m + z_{m-1} s^{m-1} + \cdots + z_1 s + z_0}{n_n s^n + n_{n-1} s^{n-1} + \cdots + n_1 s + n_0}$$

Nennerpolynom

$$G_1(s) = \frac{s+2}{s^2+5s+4}$$

```
>> G1 = tf([1 2],[1 5 4]);
```

- Pol-Nullstellen-Form der Übertragungsfunktion (SISO – System):

$$G(s) = K \cdot \frac{\prod_{j=1}^{m}(s - n_j)}{\prod_{i=1}^{n}(s - p_i)} \qquad G_2(s) = \frac{2}{(s-1)(s+2)}$$

Pole

```
>> G2 = zpk([],[1 -2],[2]);
```

Systemverstärkung

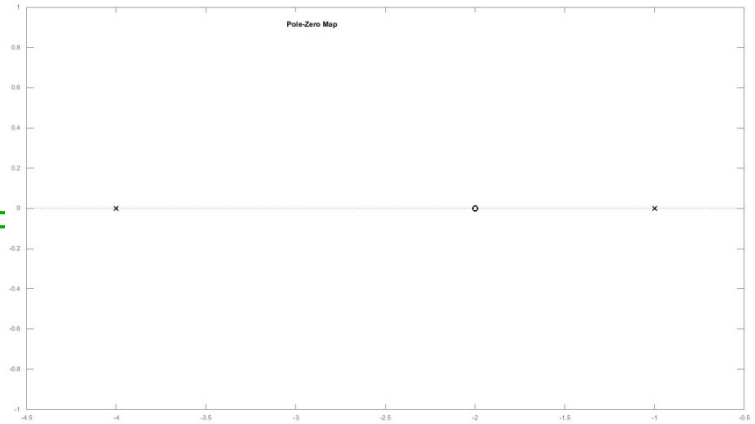
Nullstellen

- Zustandsraumdarstellung und Frequenzgangdaten: *ss*, *frd* (Literatur u. Hilfe)

Anwendung in der Regelungstechnik

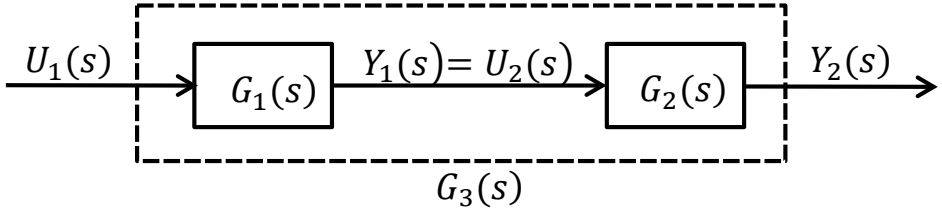
- PN-Plan:

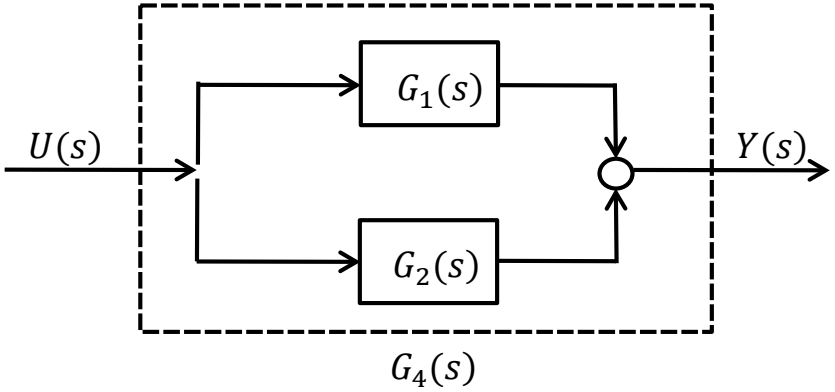
```
>> G1 = tf([1 2],[1 5 4]);
>> pzmap(G1)              % PN-Plan (OHNE
>> [p,z] = pzmap(G1);     % Vektoren von P
```



- Reihen- oder Parallelschaltung zweier Systeme:

```
>> G3=series(G1,G2)
Zero/pole/gain:
        2 (s+2)
----------------------
(s-1) (s+1) (s+2) (s+4)

>> G4=parallel(G1,G2)
Zero/pole/gain:
(s+0.522) (s^2 + 4.478s + 7.662)
--------------------------------
    (s+4) (s+2) (s+1) (s-1)
```





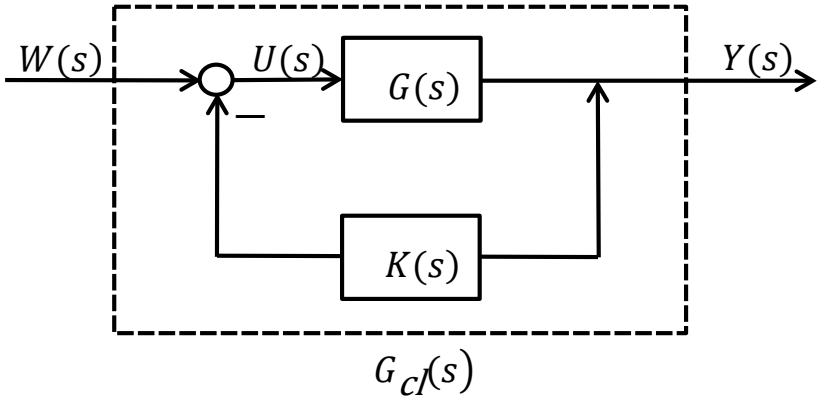
Anwendung in der Regelungstechnik

- Kürzen von Pol- und Nullstellen

```
>> G3=minreal(G3)
Zero/pole/gain:
        2
-----------------
(s-1) (s+1) (s+4) % Kürzen von Null- und Polstellen
```

- Geschlossener Kreis:

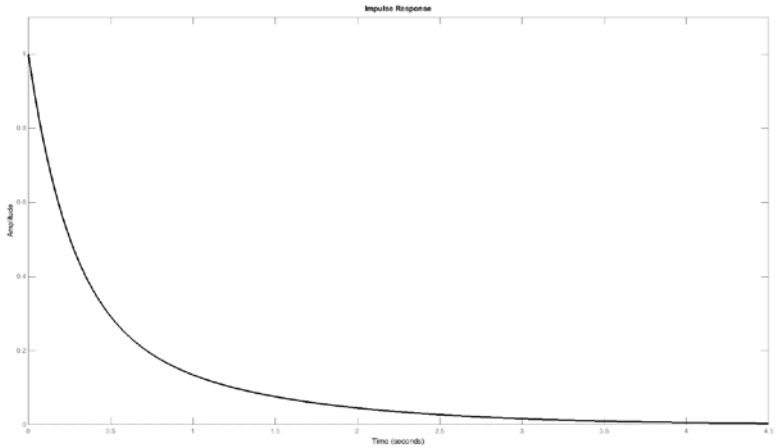
```
>> K=tf([1],[1 1])
Transfer function:
  1
-----
s + 1
>> G_cl = feedback(G1,K)
Transfer function:
    s^2 + 3 s + 2
----------------------
s^3 + 6 s^2 + 10 s + 6
```



Anwendung in der Regelungstechnik

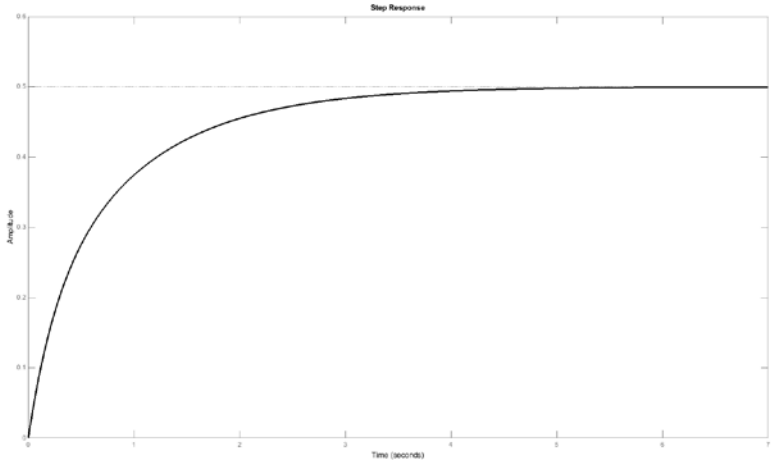
- Impulsantwort (Gewichtsfunktion) graphisch darstellen:

```
>> impulse(G1);
```



- Sprungantwort (Übergangsfunktion):

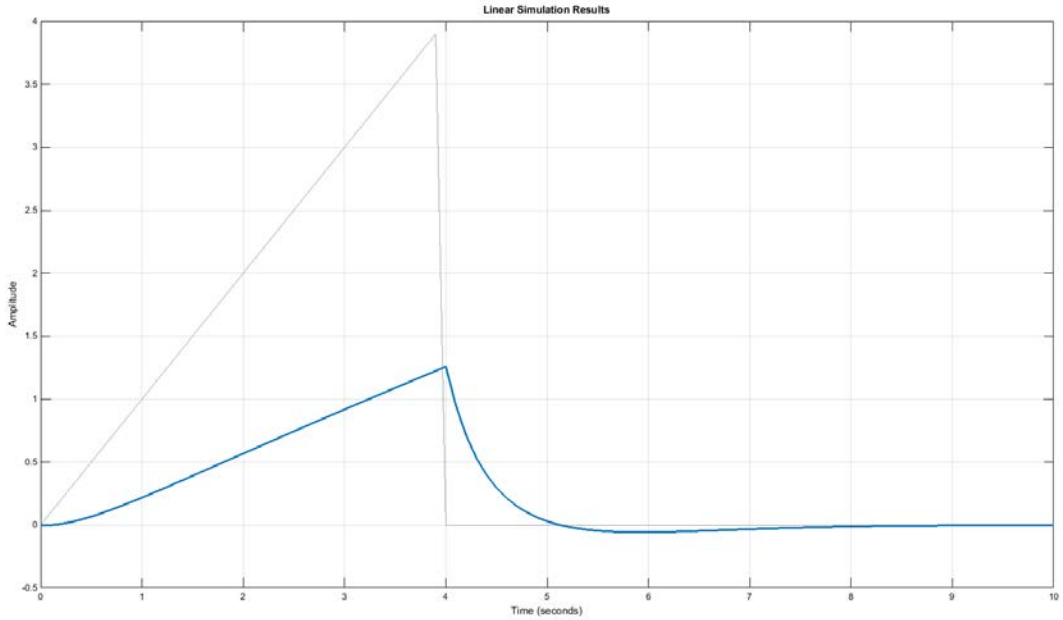
```
>> step(G1);
```



Anwendung in der Regelungstechnik

- Antwort auf beliebige Eingangssignale:

```
>> t = [0:0.1:10];
>> u = t;
>> u(41:101) = 0;
>> lsim(G_cl,u,t)
```



Anwendung in der Regelungstechnik

<u>Aufgaben:</u>

Gegeben sind die beiden Übertragungsfunktionen

$$G_1(s) = \frac{s+2}{s^2+5s+4} \qquad G_2(s) = \frac{s+2}{s^2+0.5s+1}$$

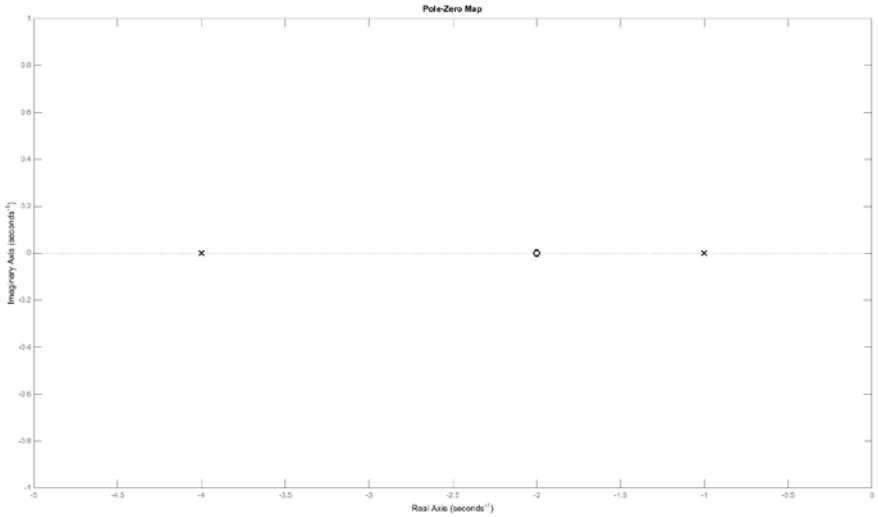
- Erstelle einen Pol/Nullstellen Plot beider Übertragungsfunktionen
- Erzeuge deren Sprung- und Impulsantwort

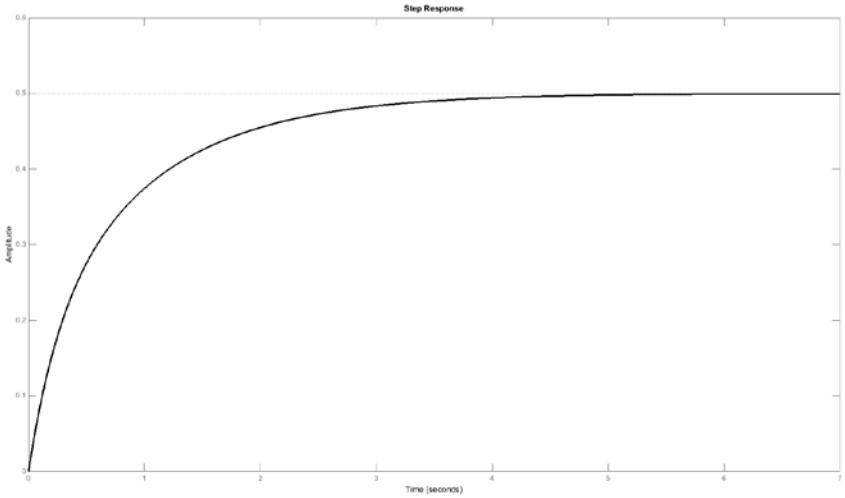
Erzeuge jeweils eine Reihen- und Parallelschaltung aus beiden Funktionen sowie eine Rückkopplung (geschl. Kreis) mit $G_2(s)$ in der Rückführung

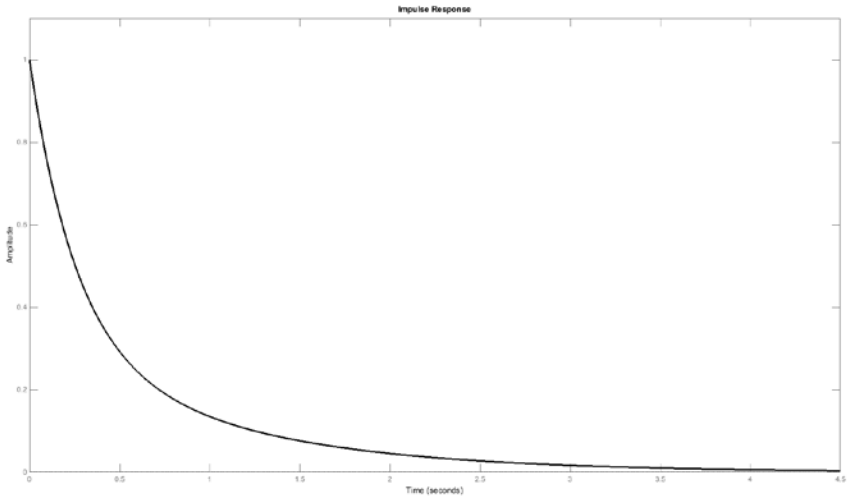
- Erstelle ebenfalls einen Pol/Nullstellen Plot der drei erhaltenen Systeme sowie deren Sprung und Impulsantwort
- Simuliere die beiden Systeme sowie deren Reihenschaltung mit folgendem Signal:
  - $t = [0{:}\,0.1{:}\,10]$ und $u = t$ (für 0 bis 50s), danach $u = 0$

Lösung:

```
>> G1 = tf([1 2],[1 5 4])

>> pzmap(G1)
>> step(G1)
>> impulse(G1)
```



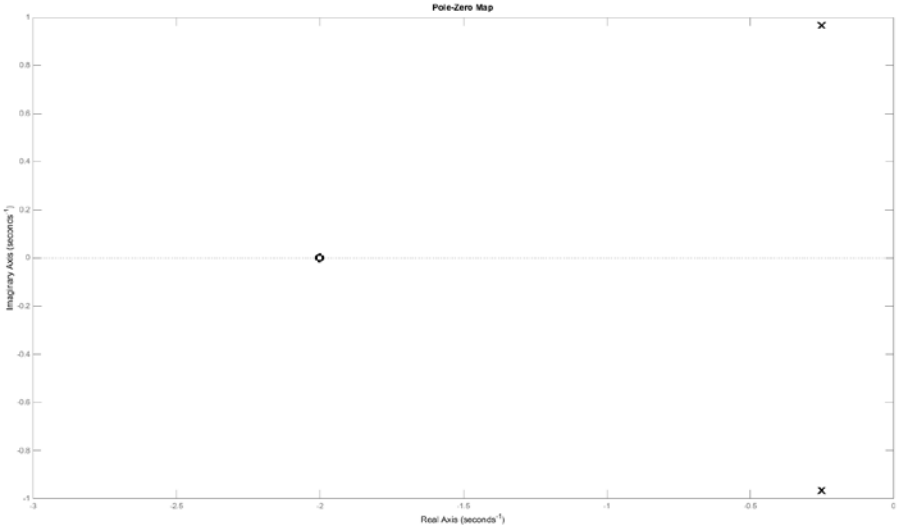


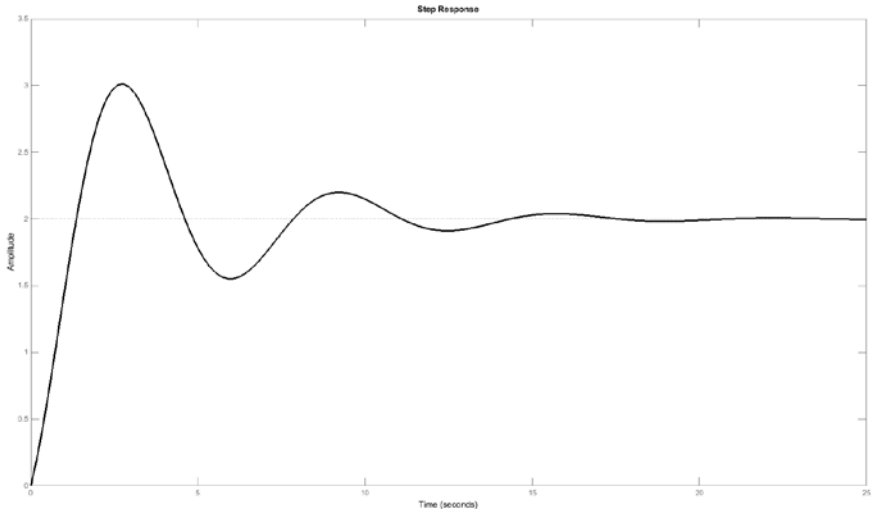


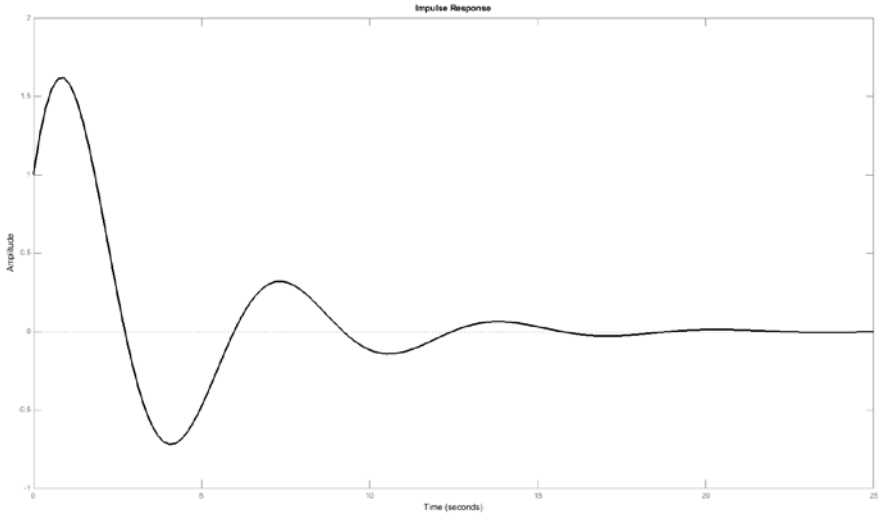
Anwendung in der Regelungstechnik

Lösung:

```
>> G2 = tf([1 2],[1 0.5 1])

>> pzmap(G2)
>> step(G2)
>> impulse(G2)
```







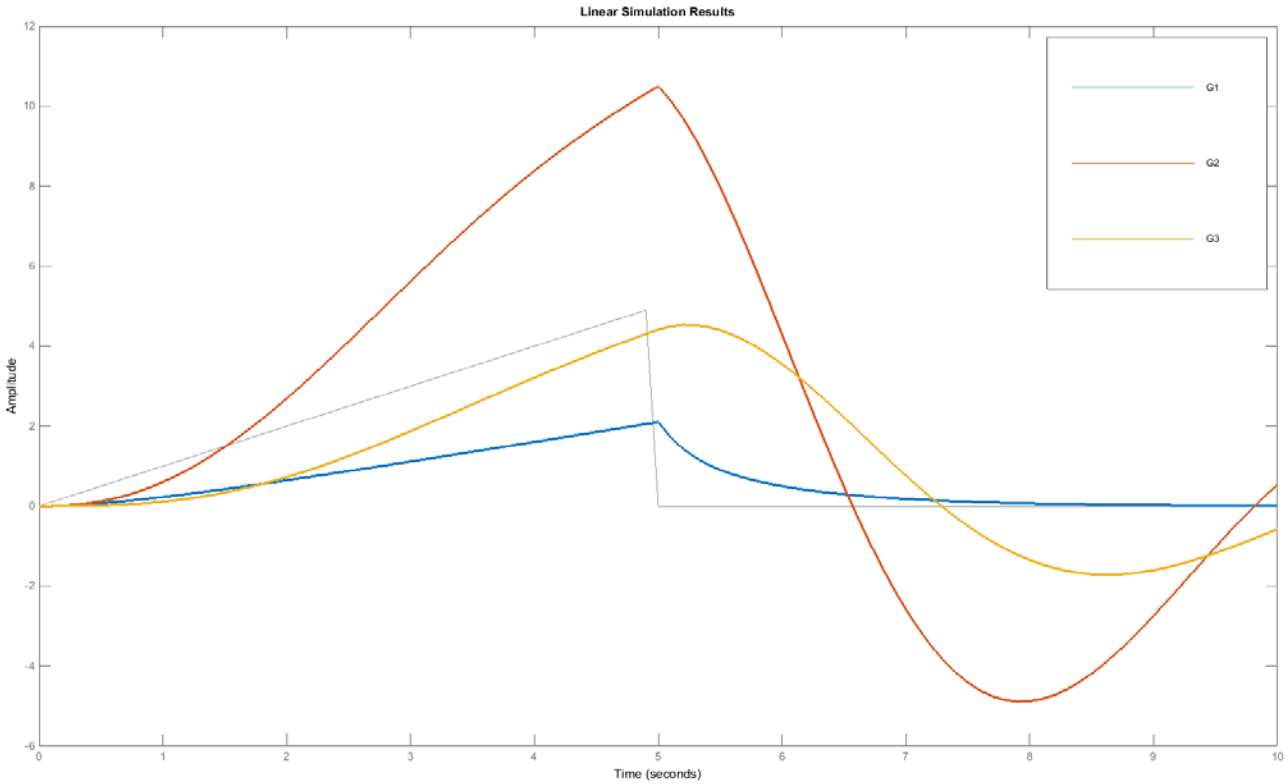
Lösung:

```
>> G3 = series(G1,G2)                    % Reihenschaltung von G1,G2
G3 =

            s^2 + 4 s + 4
  ---------------------------------
  s^4 + 5.5 s^3 + 7.5 s^2 + 7 s + 4

>> G4 = parallel(G1,G2)                  % Parallelschaltung von G1,G2
G4 =

     2 s^3 + 9.5 s^2 + 16 s + 10
  ---------------------------------
  s^4 + 5.5 s^3 + 7.5 s^2 + 7 s + 4

>> G5 = feedback(G1,G2)                  % Rückkopplung von G1 mit G2
G5 =

       s^3 + 2.5 s^2 + 2 s + 2
  ----------------------------------
  s^4 + 5.5 s^3 + 8.5 s^2 + 11 s + 8
```

Anwendung in der Regelungstechnik

Lösung:

```
t = [0:0.1:10];
u = t;
u(51:101) = 0;
lsim(G1,u,t);
hold;
lsim(G2,u,t);
lsim(G3,u,t);
```



Einführung in Simulink - Inhalt

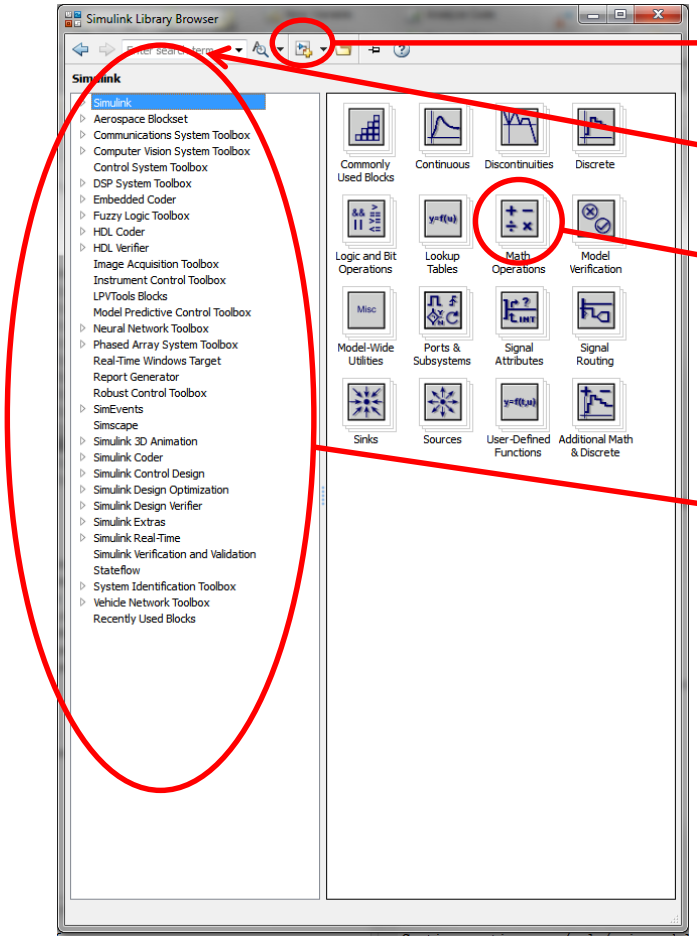
- Was ist Simulink?
- Grundsätzliches Vorgehen
- Beispiel Aufgabe
- Subsysteme
- MATLAB Funktionen in Simulink

Was ist Simulink ?

- Graphische Oberfläche zur Modellierung, Simulation und Analyse von Systemmodellen
- Simulink ist Toolbox; nutzt die vom Basismodul bereitgestellten numerischen Algorithmen.
- Simulink Erweiterungen:
  - Blocksets (z.B. Aerospace Blockset, Simscape mit SimDriveline …)
  - Extensions (z.B. Real-Time Workshop, Simulink Control Design, Stateflow ,…)
- Typische Anwendungen:
  - Codegenerierung und Rapid Control Prototyping
  - Hardware in the Loop Simulationen

Standardansicht

- Aufruf im Command Window: `>> simulink`



Neues Simulink Modell erstellen

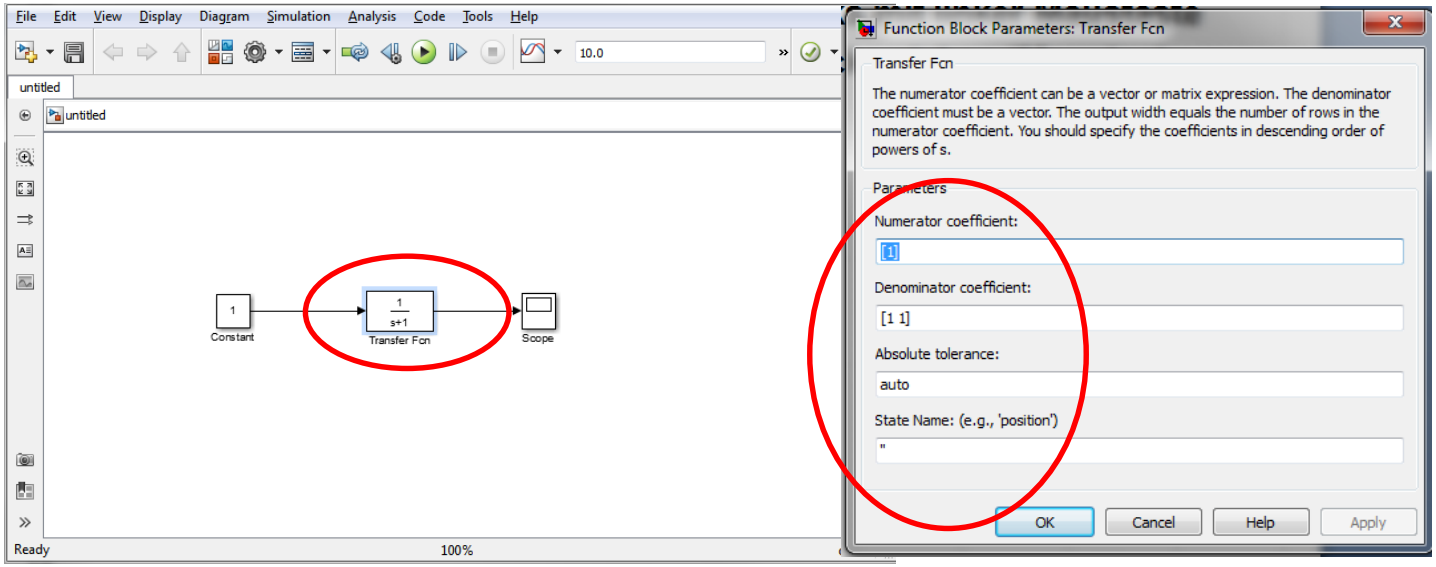
Nach Simulink Block suchen

Einzelne Simulink Bibliothek

Simulink Library Browser (verfügbare Bibliotheken, Blocksets, Extensions)

Vorgehen: 1. Simulink-Modell erstellen

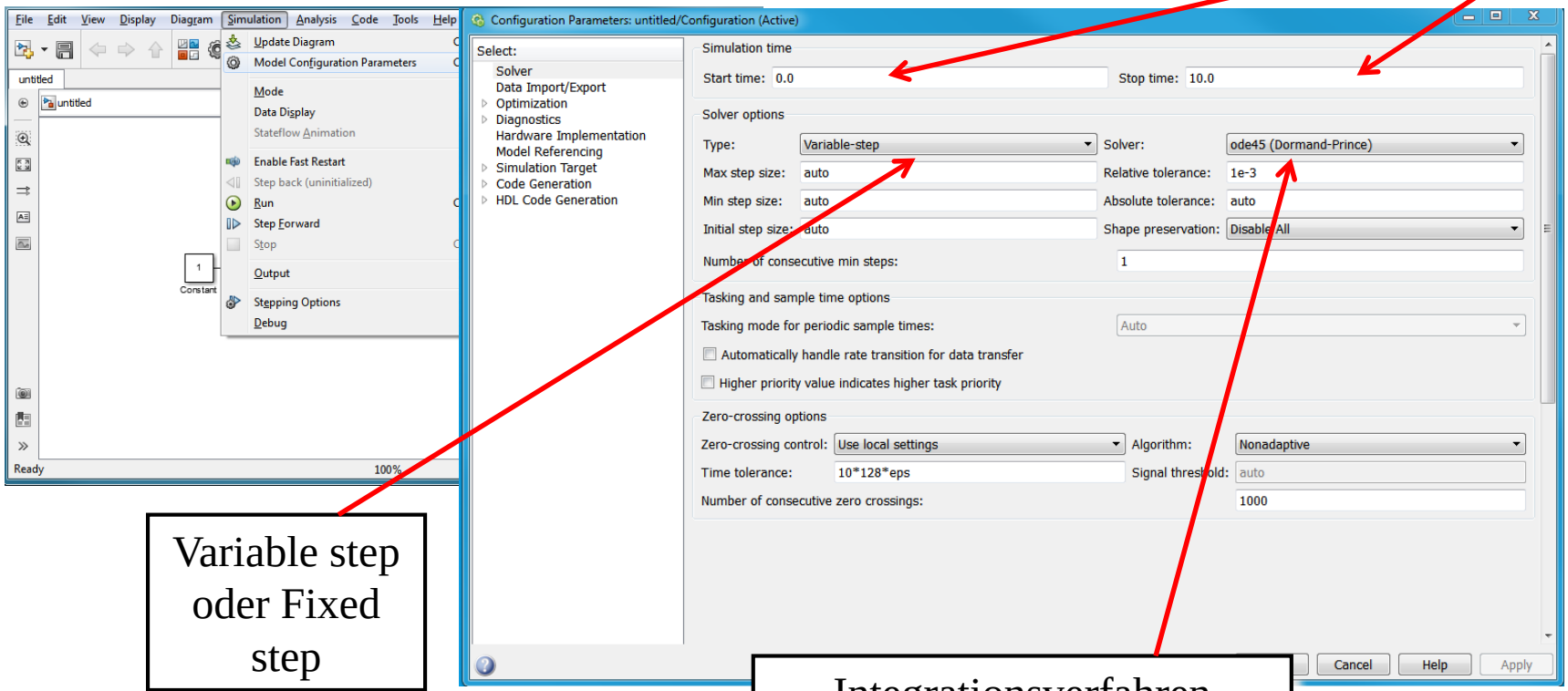
- Platzieren und Verbinden der Simulink-Blöcke:
  - Simulink-Block per Drag-and-Drop aus Bibliothek in Modellfenster ziehen.
  - Blöcke verbinden: Ausgang des ersten Blocks mit linker Maustaste anklicken und zum Eingang des zweiten Blocks ziehen (Maustaste gedrückt halten).
  - Einstellen der Blockparameter: Doppelklick auf Simulink-Block.



Vorgehen: 2. Simulationsparameter einstellen

- Grundlegende Einstellungen zur Simulation:
  - Simulation → Model Configuration Parameters…

Zeit-intervall



Variable step oder Fixed step

Integrationsverfahren

Vorgehen: 2. Simulationsparameter einstellen

- Grundlegende Einstellungen zur Simulation:
  - Integrationsverfahren mit fester Schrittweite:

  → Definierte Rechenzeit, notwendig für Echtzeitanwendungen.

  - Integrationsverfahren mit variabler Schrittweite:

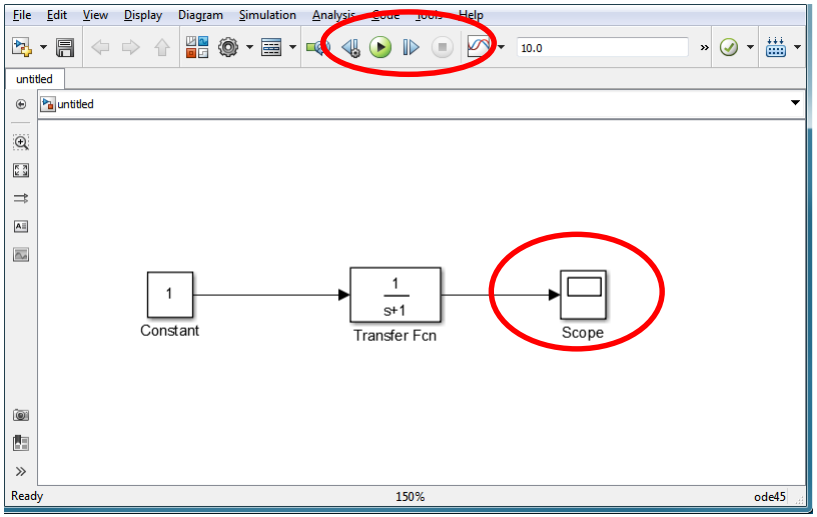
  → Genauigkeit, Geschwindigkeit, Fehlerüberwachung und Erkennung von Nulldurchgängen.

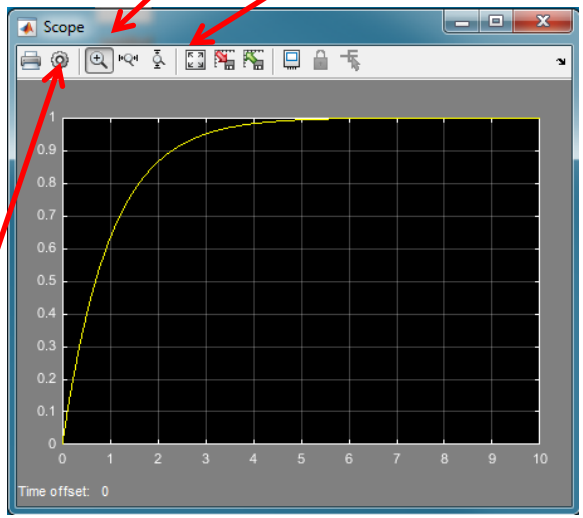
- <u>Tipp:</u> Zunächst Simulation mit Standard-Einstellungen, bei Bedarf an Systemdynamik anpassen

Vorgehen: 3. Simulieren und Visualisieren

- Start der Simulation:
  - Start mit ▶, Pause mit ❚❚ und Ende mit ■. (Simulation->Start)

- Einfache Visualisierung mit Scope (Doppelklick):





Zoom

Autozoom

Parameter

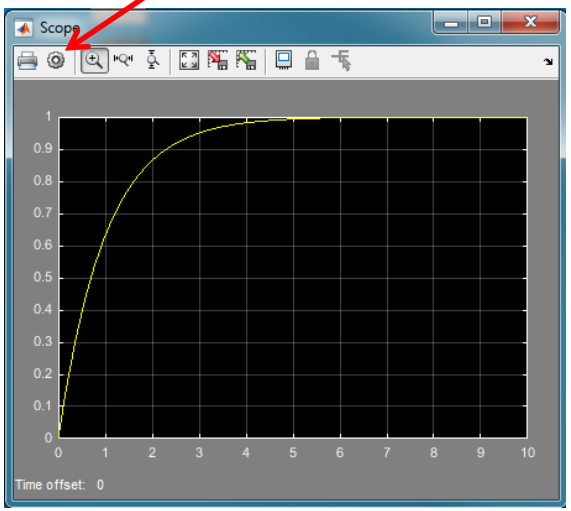
Vorgehen: 4. Auswerten und Weiterverarbeiten

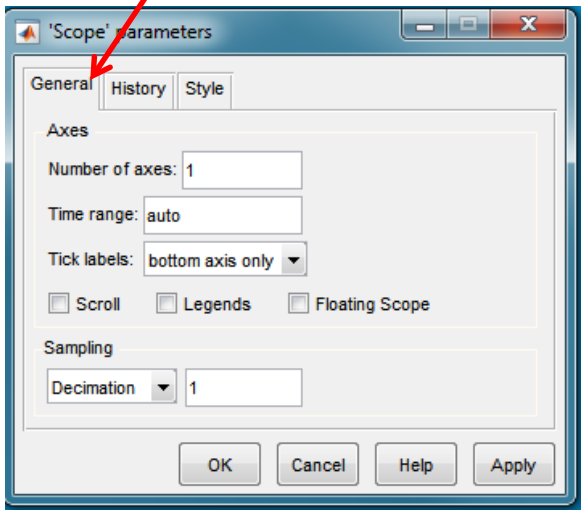
- Ergebnisse der Simulation auswerten und weiterverarbeiten:

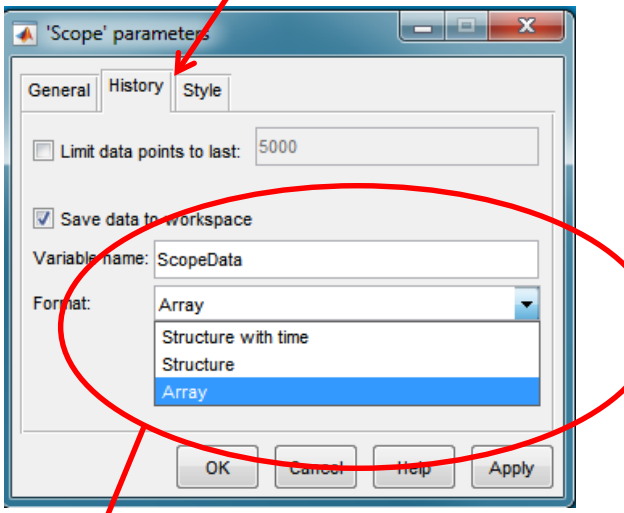
Parameter

General

Data History



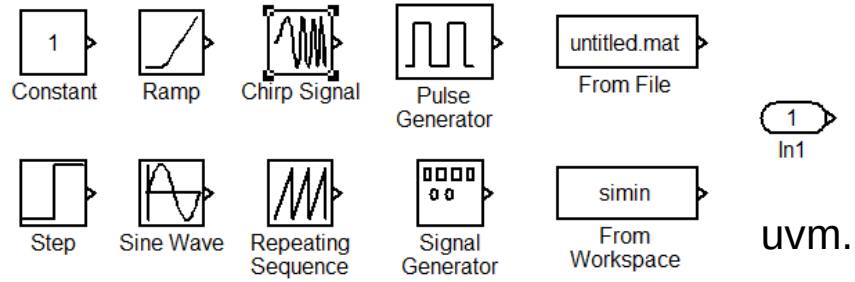




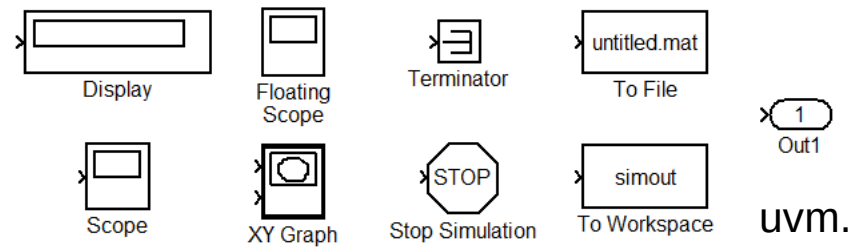
Daten als Structure oder Array in MATLAB Workspace

Wichtige Simulink Bibliotheken

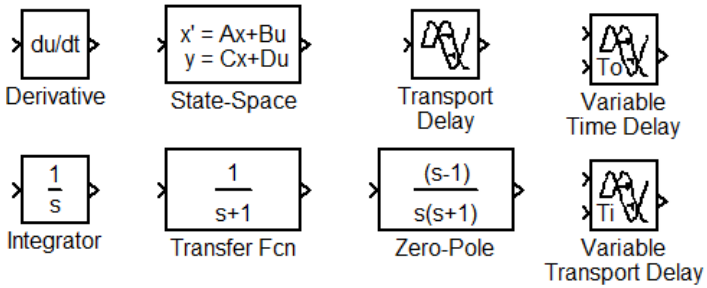
- **Sources:**
  - Eingangssignale erzeugen
  - Einlesen von Daten aus dem MATLAB Workspace
  - Einlesen von Daten aus Dateien



- **Sinks:**
  - Visualisieren von Signalen
  - Schreiben von Daten auf den MATLAB Workspace
  - Schreiben von Daten in Dateien

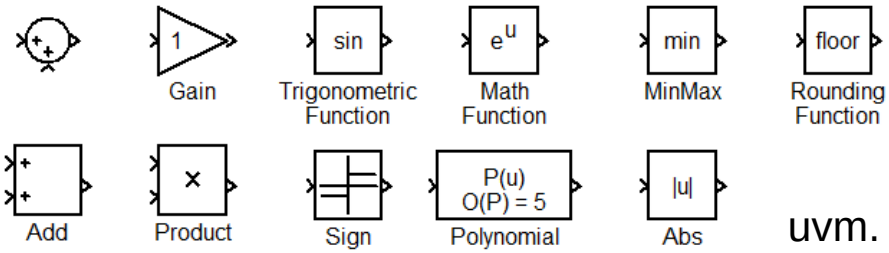


- **Continuous:**
  - zeitkontinuierliche Systeme
  - Totzeiten

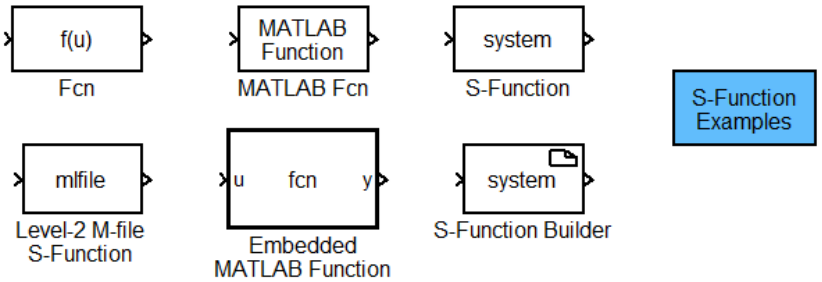


Weitere Simulink Bibliotheken, Blocksets

- **Bibliothek Math Operations:**
  - Arithmetische Operationen
  - Mathematische und trigonometrische Funktionen



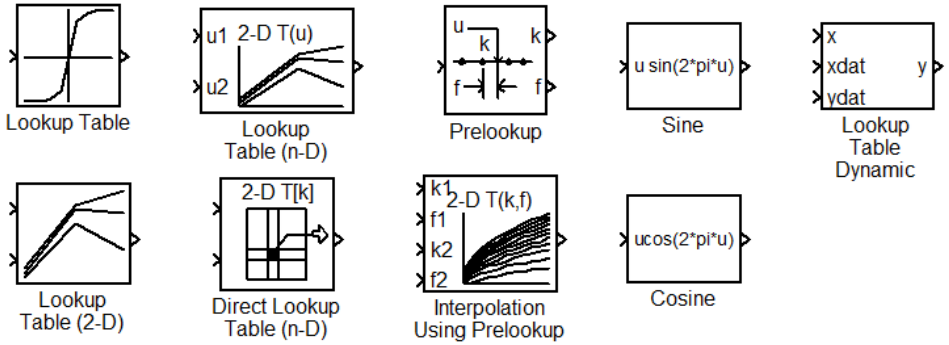
- **Bibliothek User Defined Functions:**
  - Frei programmierbare Funktionen
  - S-Functions



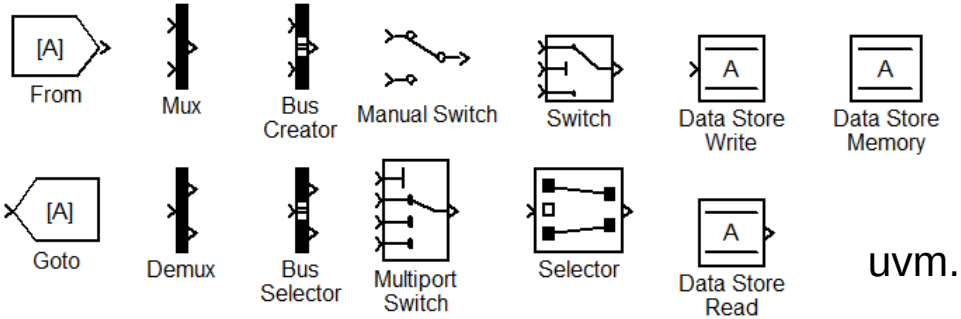
Weitere Simulink Bibliotheken, Blocksets

- **Bibliothek Lookup Tables:**

  Approximation von Kennlinien und Kennfeldern mit diskreten Werten und unterschiedlichen Interpolationsverfahren

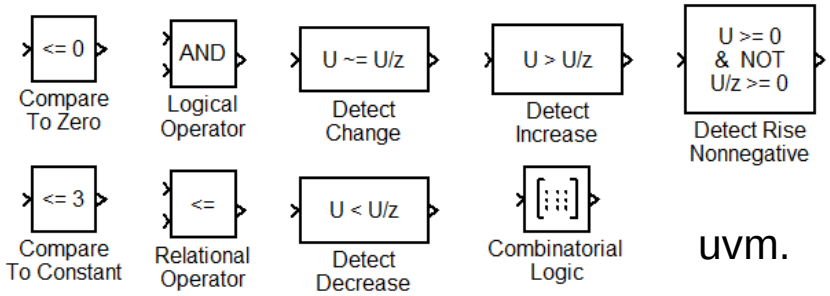


- **Bibliothek Signal Routing:**
  - Verknüpfung und Auswahl von Signalen
  - Datenspeicher-Management



Weitere Simulink Bibliotheken, Blocksets

- **Bibliothek Logic and Bit Operations:**
  - Logische Operationen
  - Operationen auf Bitebene,
  - Signalüberwachung

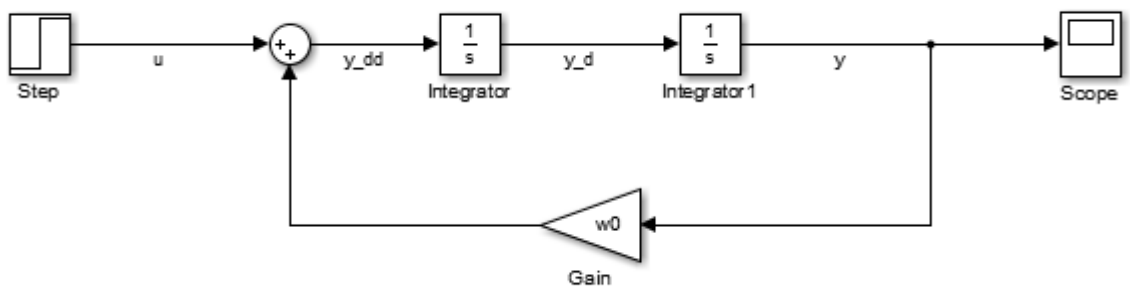


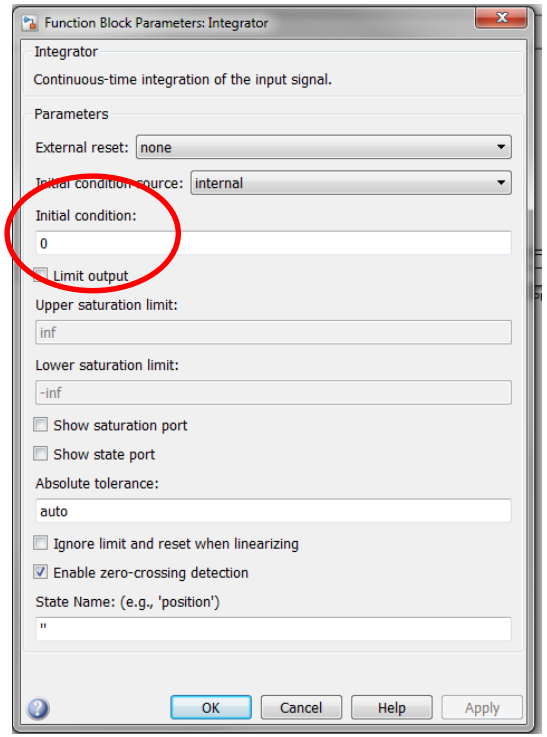
uvm.

Beispiel – Simulation einer DGL

Es soll die folgende DGL in Simulink programmiert werden:

$$\ddot{y}(t) - \omega_0 \cdot y(t) = u(t)$$

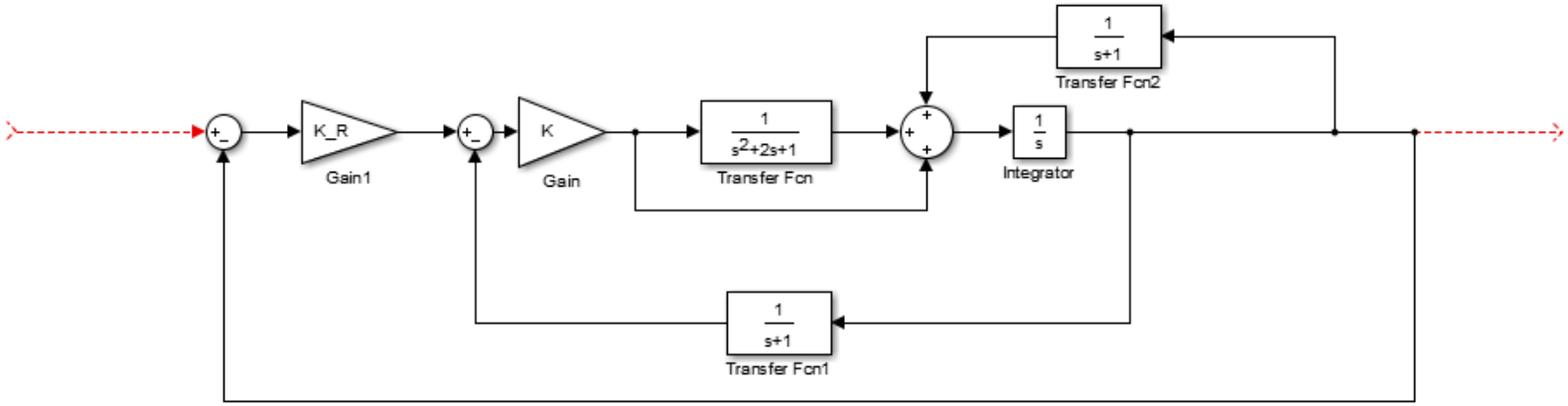




- Anfangswerte für die zeitabhängigen Größen werden am jeweiligen Integrator vorgegeben
- Eingegebener Wert entspricht dem Anfangswert des Ausganges

Beispiel – Simulation eines Regelkreises

Beispiel eines Regelkreises:



- Einfaches Aufbauen von Blockschaltbildern
- Kein vorheriges Zusammenfassen notwendig
- Direkte Simulation und damit Testen von Reglern möglich
- Schnelles anpassen bzw. ändern

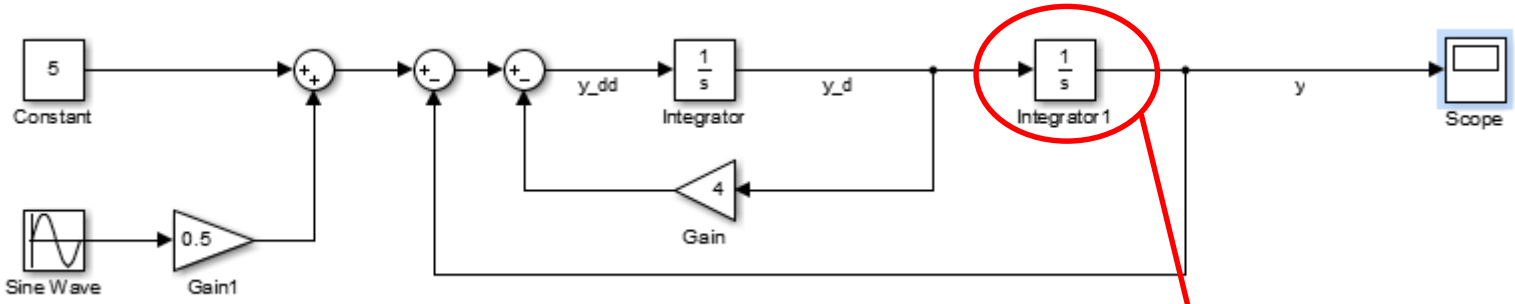
Aufgaben

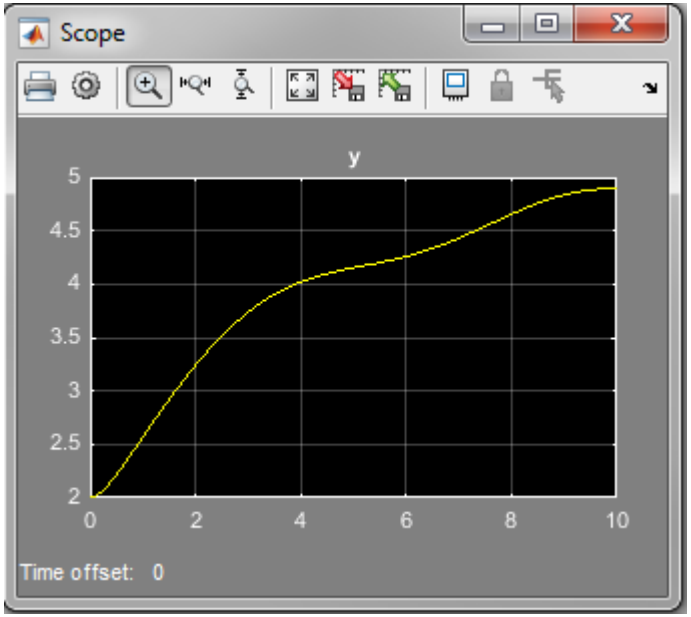
<u>Aufgaben:</u>

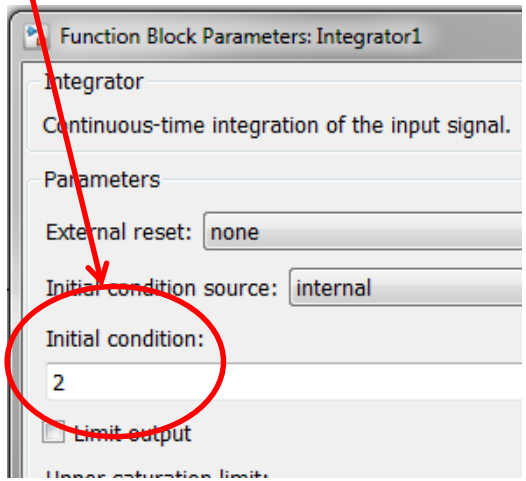
- Erstellen zweier Simulink-Modelle:
  - Differentialgleichung $\ddot{y}(t) + 4 \cdot \dot{y}(t) + y(t) = u_1(t) + 0.5 \cdot u_2(t)$
  - Eingangssignale:
    - $u_1 = 5$
    - $u_2 = \sin(t)$
    - Startwerte: $\mathrm{y}(0) = 2, \ddot{y}(0) = \dot{y}(0) = 0$

  - Rampe am Eingang der Strecke 1/(1+s)
  - Ausgang soll visualisiert werden
  - Rampe: $u(0) = -1$, Rampe beginnt z.Zt. 1, hat Anstieg 2
- Simulationsparameter:
  - Zeitintervall [0,10]
  - Restliche Einstellungen auf den Standardeinstellungen belassen
- Simulieren, Visualisieren

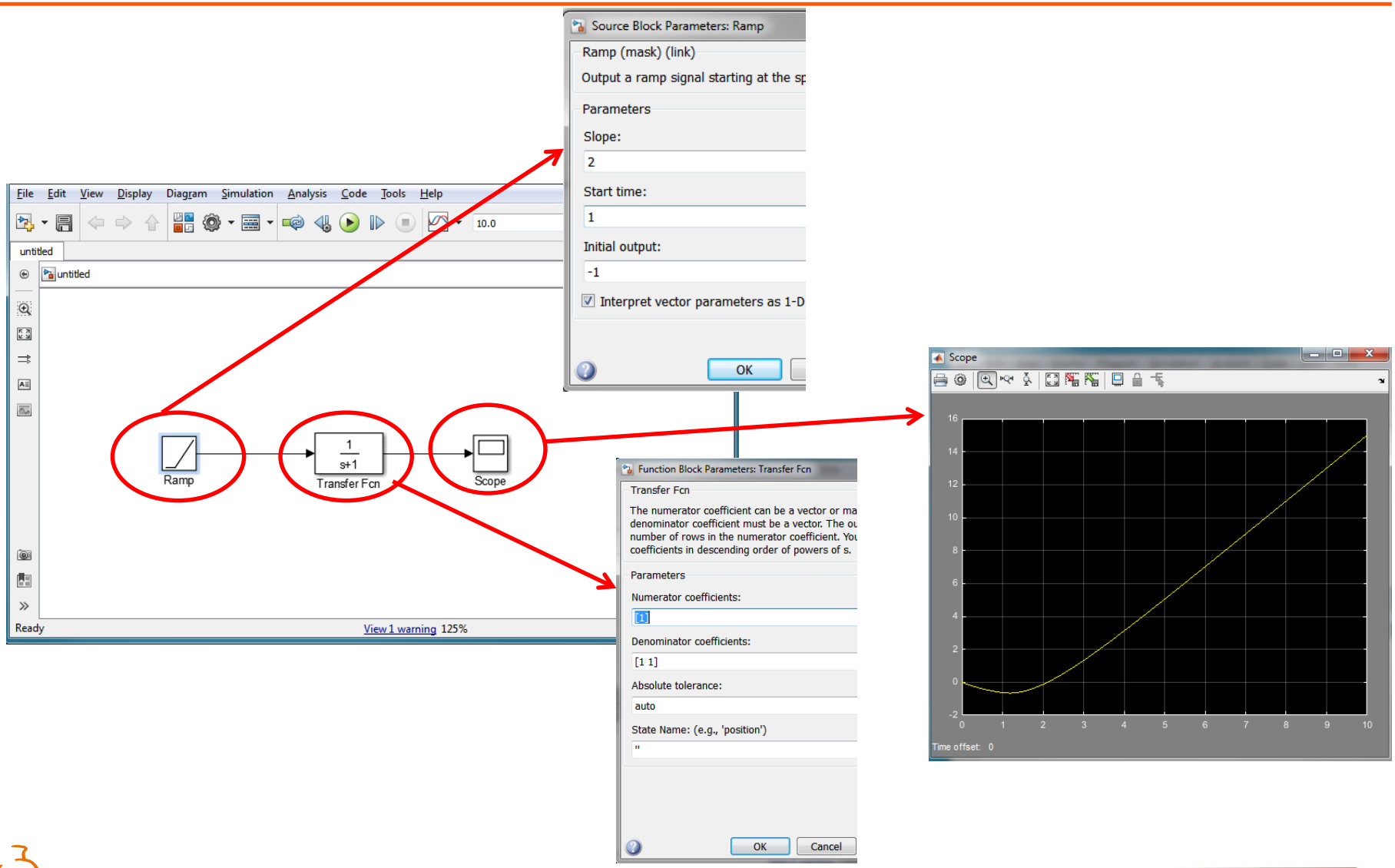
Lösung


5
Constant
y_dd
1
s
Integrator
y_d
1
s
Integrator1
y
Scope
4
Gain
0.5
Sine Wave
Gain1

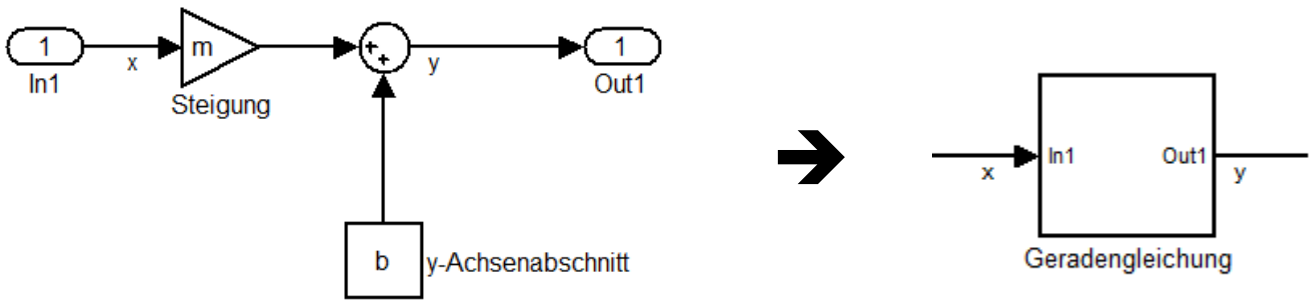

Scope
y
5
4.5
4
3.5
3
2.5
2
0
2
4
6
8
10
Time offset: 0


Function Block Parameters: Integrator1
Integrator
Continuous-time integration of the input signal.
Parameters
External reset: none
Initial condition source: internal
Initial condition:
2
Limit output

Lösung


Source Block Parameters: Ramp
Ramp (mask) (link)
Output a ramp signal starting at the sp
Parameters
Slope:
2
Start time:
1
Initial output:
-1
Interpret vector parameters as 1-D
OK
File Edit View Display Diagram Simulation Analysis Code Tools Help
10.0
untitled
Ramp
1
s+1
Transfer Fcn
Scope
Ready
View 1 warning 125%
Function Block Parameters: Transfer Fcn
Transfer Fcn
The numerator coefficient can be a vector or ma
denominator coefficient must be a vector. The ou
number of rows in the numerator coefficient. You
coefficients in descending order of powers of s.
Parameters
Numerator coefficients:
[1]
Denominator coefficients:
[1 1]
Absolute tolerance:
auto
State Name: (e.g., 'position')
''
OK
Cancel
Scope
Time offset: 0

- **Verwendung von Subsystemen (1/2):**
  - Mehrere Blöcke können zu einem neuen Simulink-Block zusammengefasst werden:
  - *Edit → Create Subsystem*



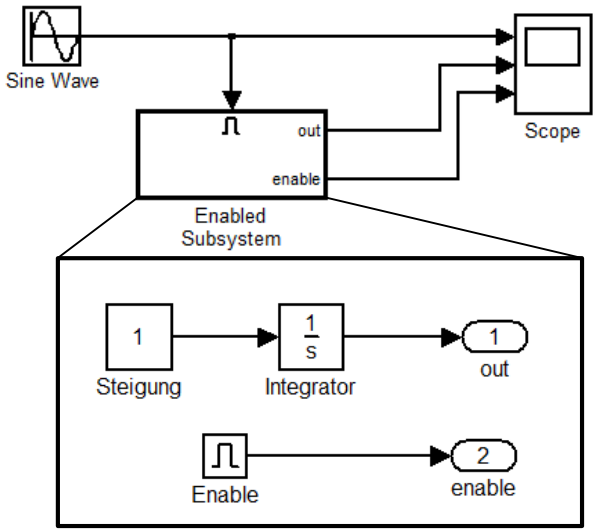
  - Übersichtlichkeit
  - Effizienz (Wiederverwendung getesteter Strukturen...)
  - Parametrisierung von komplexen Teilsystemen mit rechter Maustaste: *Mask Subsystem → Parameters*
  - Subsysteme, die oft benötigt werden → eigene Bibliothek !

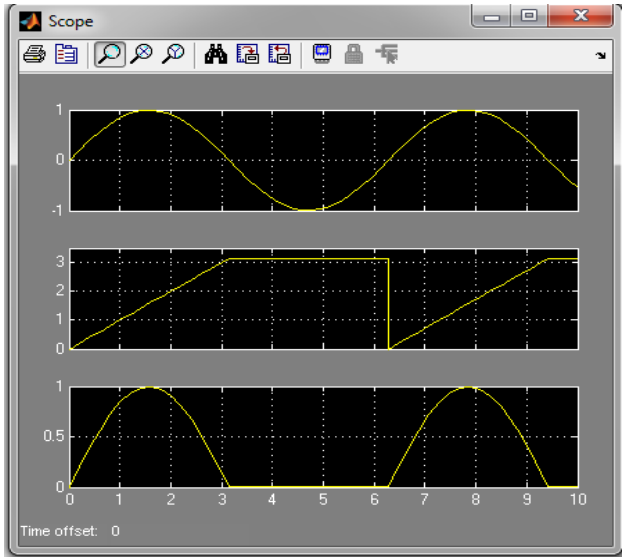
- **Verwendung von Subsystemen (2/2):**
  - Ausführung des Subsystems kann durch Steuersignal bestimmt werden, abhängig von Bedingung(en):

→ *Enabled Subsystems, Triggered Subsystems*

→ *Control Flow Subsystems, für Schleifen und Verzweigungen*

- Beispiel: $y_1(t) = sin(t), y_2(t) = t \cdot 1(t)$ für $sin(t) > 0$



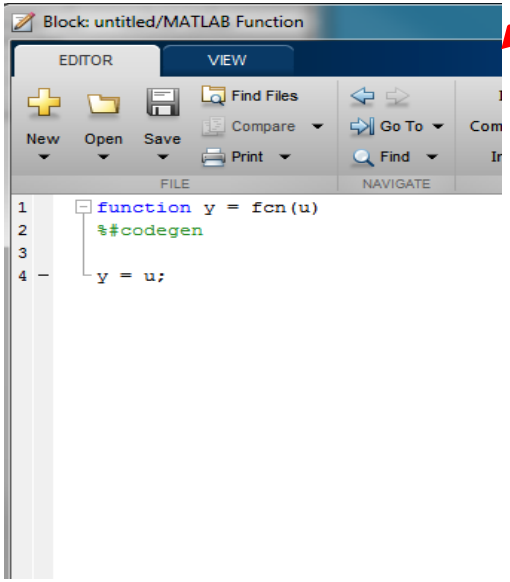


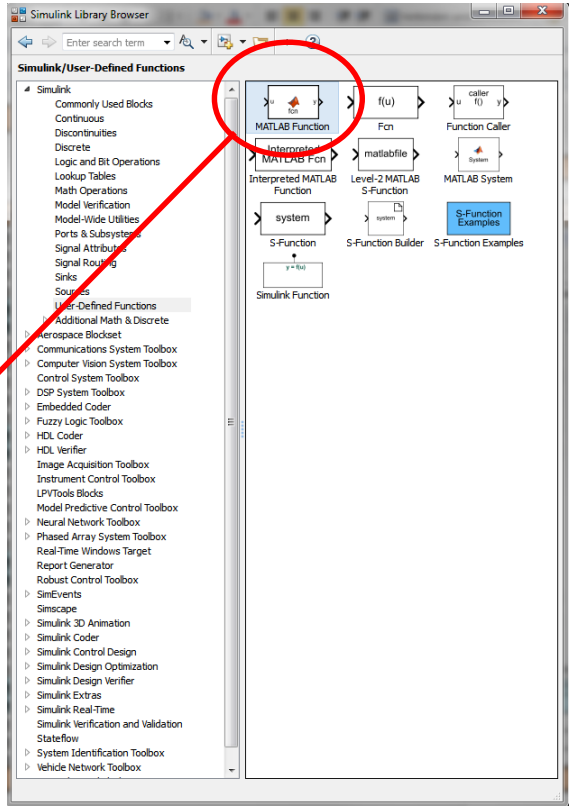
MATLAB-Functions

- **MATLAB Function:** frei programmierbarer Simulink-Block
- Programmierung analog zur *function* in Matlab
- Ermöglicht auch das Aufrufen von bereits bestehenden Matlab-Funktionen
- Einfachere Programmierung als mit elementaren Simulink-Blöcken.
- Geringere Rechenzeit als mit elementaren Simulink-Blöcken.
- Bei komplizierten Rechnungen häufig einfacher umsetzbar
- Weitere Anwendung: Systembeschreibung liegt als Code vor (zB von einem Industriepartner), aber nicht als Modell aus elementaren Blöcken.

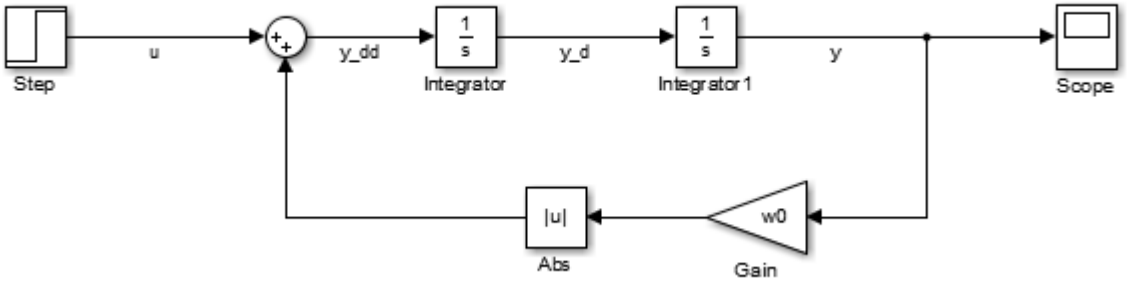
MATLAB-Functions

- **Wie erstelle ich eine MATLAB-Function?**
  - Block „MATLAB-Function“
  - User Defined Functions
  - Erstellung analog zur Funktion in Matlab
  - Definition von Ein- und Ausgängen
  - Programmieren von beliebigen Rechenoperationen

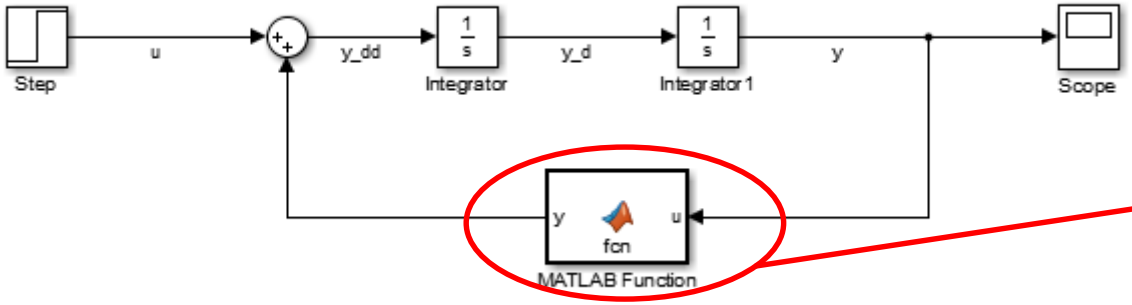


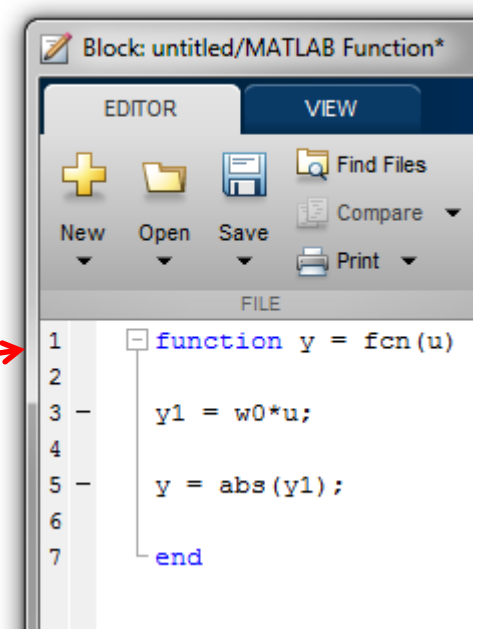


MATLAB-Functions Beispiel



Ersetzen der Rückführung durch MATLAB-Function





- **<u>Aufgabe 1:</u> „Schiefer Wurf“**

Simulieren Sie die Flugbahn einer punktförmigen Masse M unter Einfluß der Schwerkraft, die bei $t = 0$ im Punkt $(0, h)$ mit der Geschwindigkeit $(v_{x0}, v_{z0})^T$ startet

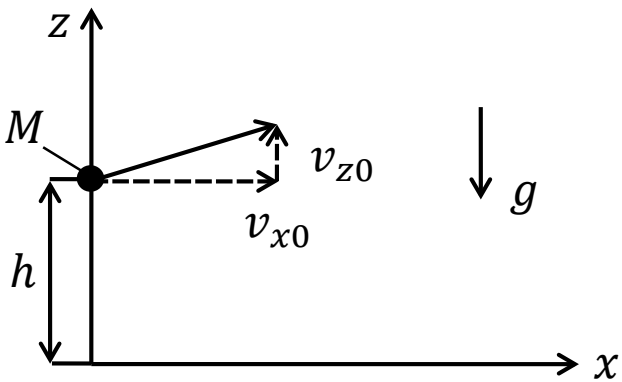


Abbildung 1: Schiefer Wurf

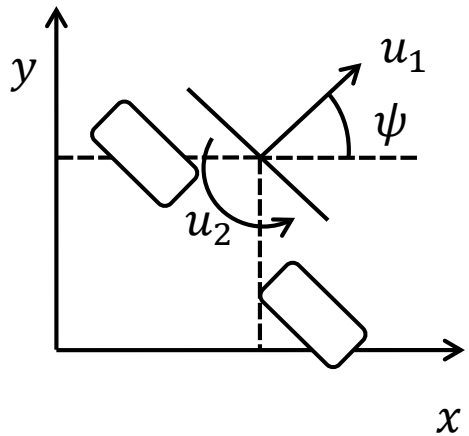
Tipp: $\dot{x} = v_{x0}$,
$\ddot{z} = -g$.

Vernachlässigen Sie alle Reibungs- und Kontakteffekte.

$M = 5kg, h = 5m, g = 9.81\ \frac{m}{s^2},\ \ (v_{x0}, v_{z0}) = (15{,}10)\ \frac{m}{s}$

- **<u>Aufgabe 2:</u> Fahrdynamik eines Roboters**

Es wird das folgende Modell eines Roboters betrachtet:



$$\dot{x} = u_1 \cdot cos(\psi),$$

$$\dot{y} = u_1 \cdot sin(\psi),$$

$$\dot{\psi} = u_2.$$

$$(x_0, y_0, \psi_0)^T = (0,0,0)^T.$$

Wählen Sie $u_1 = 5, u_2 = 0.1 \cdot \sin(t)$