



6

Modellierung zeitkont. Systeme

am Beispiel von MatLab

6.1 Kurze Übersicht zu MatLab

6.2 Einführung in MatLab

6.3 Modellierung mit Simulink

6.4 Modellierung hybrider Systeme mit Stateflow



6.1.1 Matlab – Simulink – Stateflow

Matlab

- Programmumgebung für numerisches Rechnen
- MatLab : **Mat**ritzen **Lab**oratorium
- sehr weit verbreitet
- sehr viele Toolboxes : Neuronale Netze, Fuzzy, Bildverarbeitung, Robotik
Regelungstechnik, Zustandsautomaten, usw.

Simulink

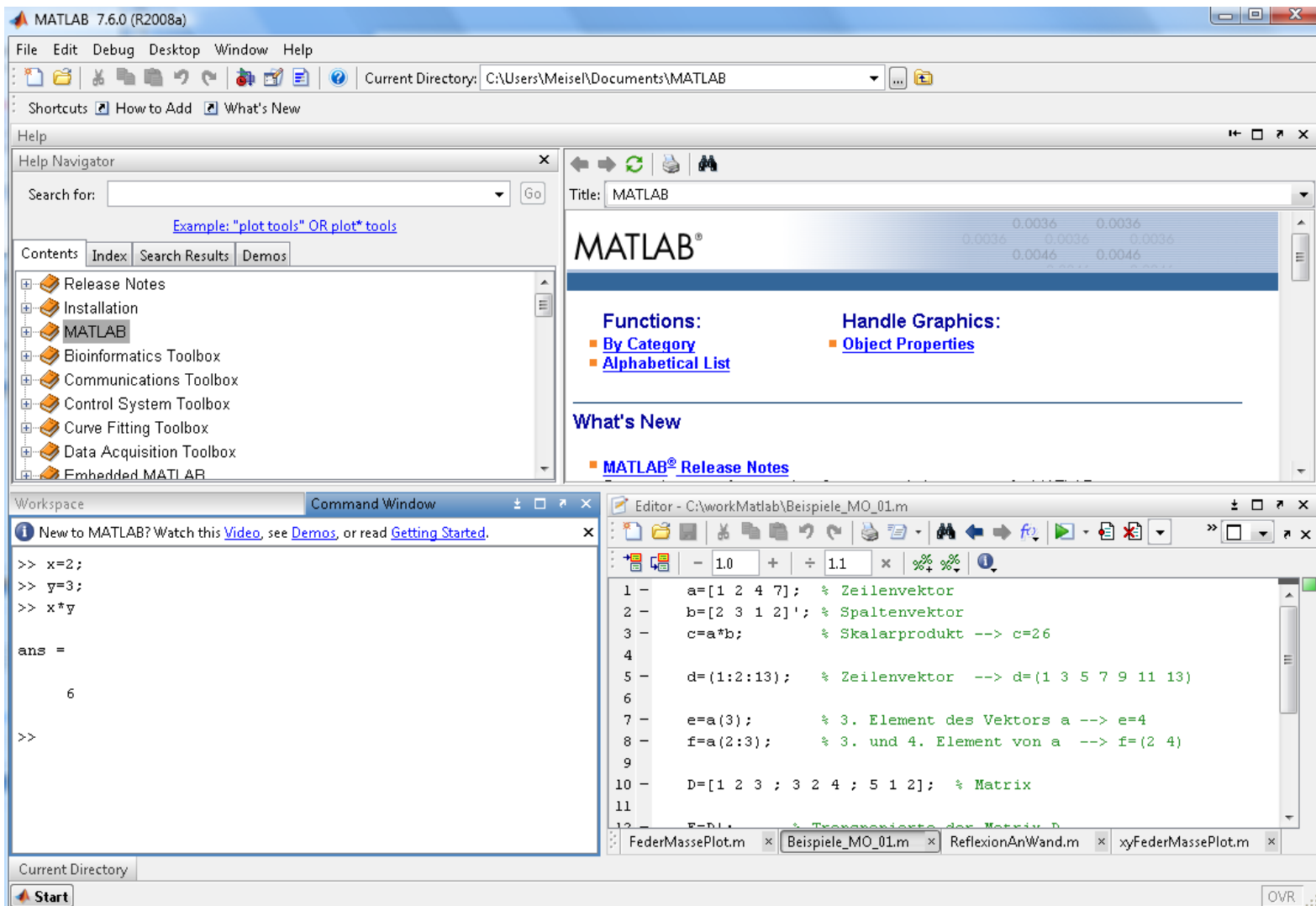
- grafisch interaktive Benutzeroberfläche zur Modellierung und Simulation dynamischer Systeme

Stateflow

- grafisch interaktive Modellierung von Zustandsautomaten
- eingebunden in Simulink



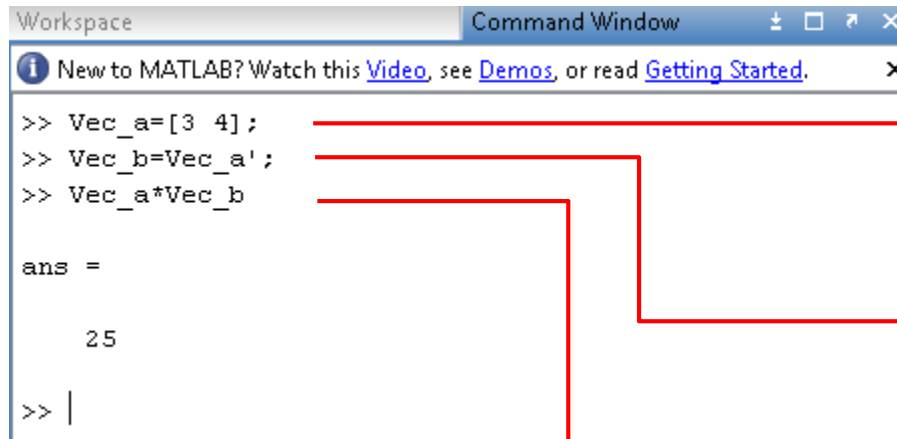
6.1.2 Matlab – Desktop





6.1.2.1 Command Window

Direkte Eingaben und Berechnungen



```
>> Vec_a=[3 4];
>> Vec_b=Vec_a';
>> Vec_a*Vec_b

ans =

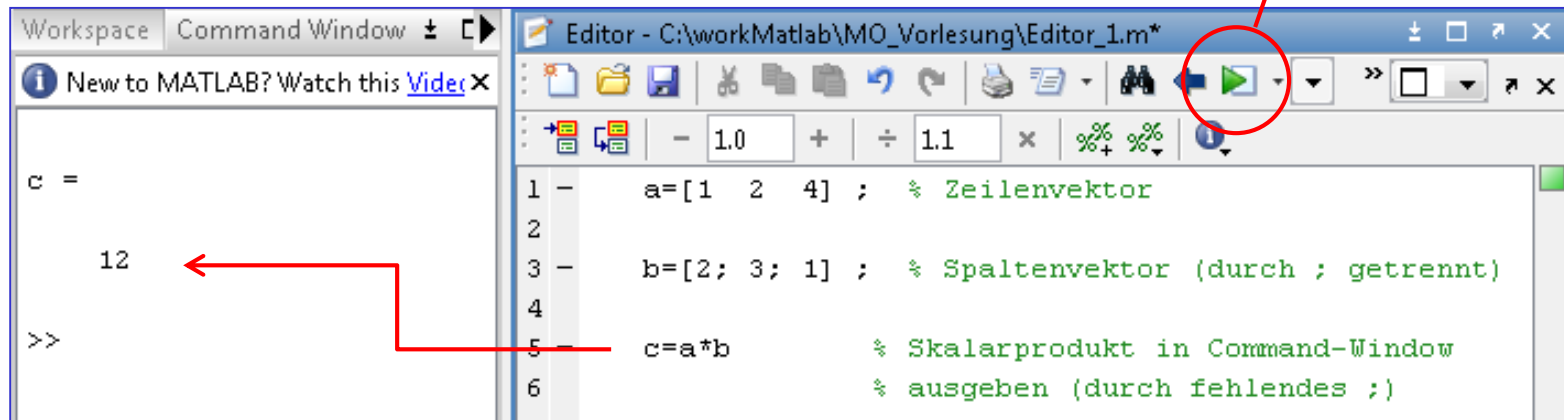
    25

>> |
```

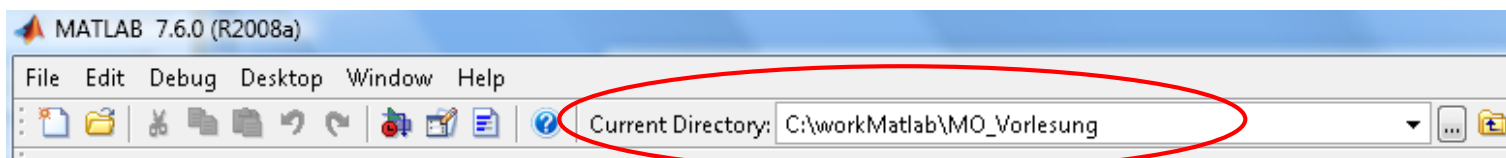
- Vec_a ist ein Zeilenvektor
- durch ; wird Ausgabe unterdrückt
- Vec_b ist die Transponierte von Vec_a (durch ')
- Skalarprodukt berechnen und ausgeben (da ; fehlt)

6.1.2.2 Editor

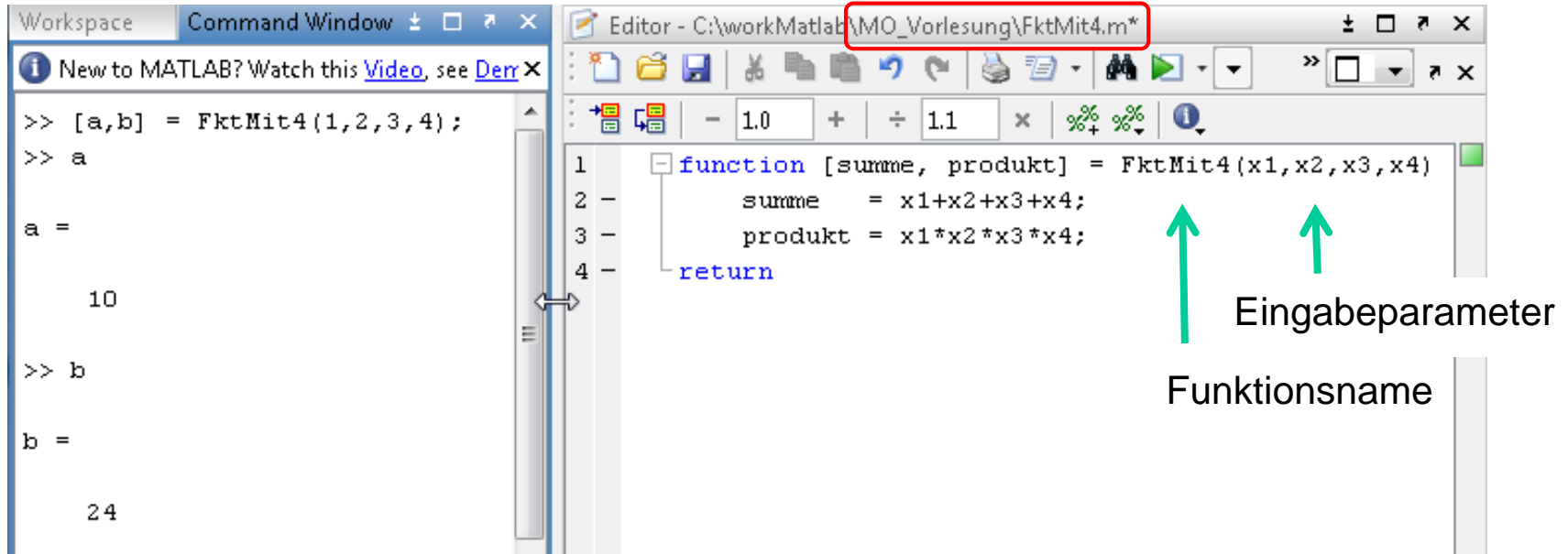
Eingeben und starten von Skripten



- Vorteil des Editors: die Session kann abgespeichert werden und das Skript kann mit geänderten Werten erneut berechnet werden
- Vor dem Abspeichern (.m-Datei) muss das „Current Directory“ gesetzt werden



Definition eigener Funktionen



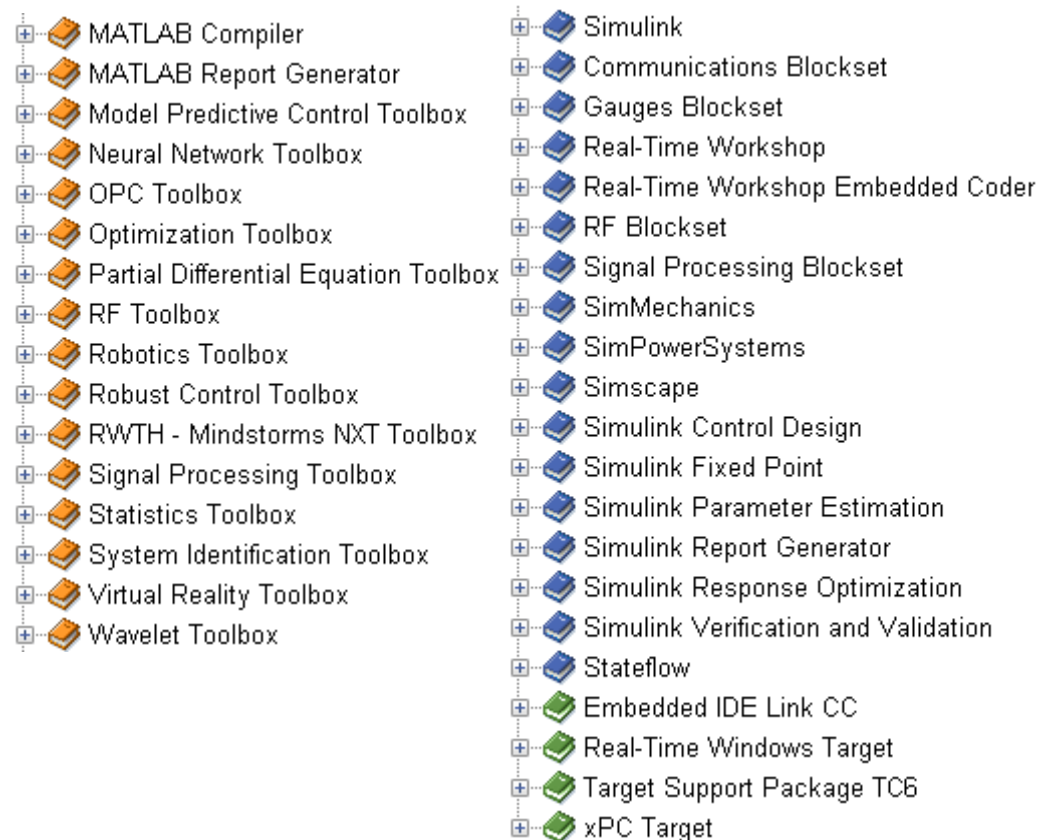
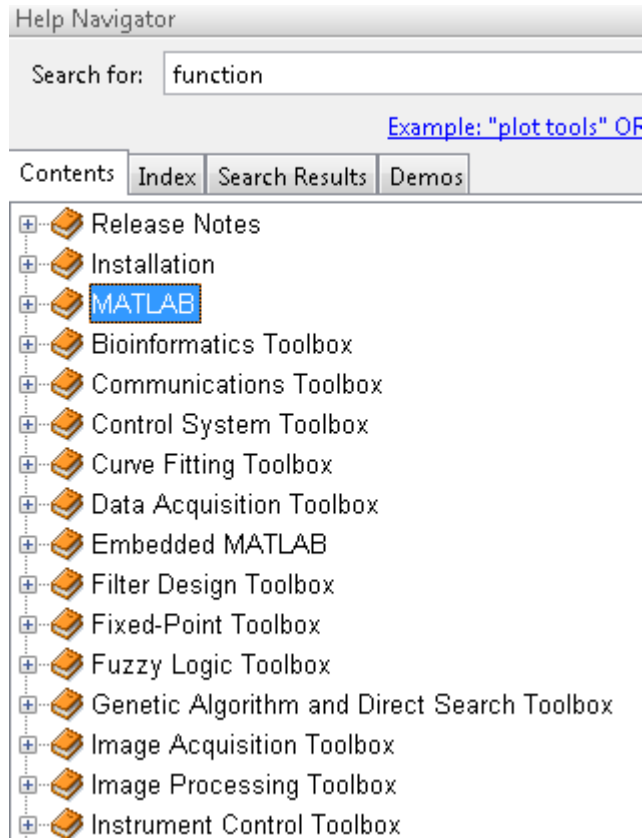
The image shows the MATLAB environment. On the left, the Command Window displays the execution of the function `FktMit4` with inputs 1, 2, 3, and 4, resulting in `a = 10` and `b = 24`. On the right, the Editor window shows the definition of `FktMit4.m`. The function signature `function [summe, produkt] = FktMit4(x1,x2,x3,x4)` is highlighted with a red box. Two green arrows point to the input parameters `x1, x2, x3, x4` in the signature, labeled "Eingabeparameter". Another green arrow points to the function name `FktMit4`, labeled "Funktionsname". The function body calculates the sum and product of the inputs and returns them.

```
function [summe, produkt] = FktMit4(x1,x2,x3,x4)
    summe = x1+x2+x3+x4;
    produkt = x1*x2*x3*x4;
    return
```

- **Dateiname** und der **1. Funktionsname** müssen gleich sein (hier `fktmit4.m`)
- Dateiendung ist `.m` (M-File)
- Beim Funktionsaufruf wird die Funktion im „Current Directory“ und im „Matlab-Pfad“ gesucht.



6.1.3 Matlab – Toolboxes





6.1.4 Matlab – Datenformate

- Einfache Datentypen für skalare Werte und Matrizen:
 - single/double, 8-/16-/32-bit signed/unsigned integer, logical, char
 - Standarddatentyp ist double

```
>> a=int8(5);
>> b=true;
>> c=15;
```

```
>> d='a';
>> e=uint32(123);
>> f=[1 2 3; 4 2 1];
```

- Strukturen

```
>> Prof.Name='Meisel';
>> Prof.Groesse=1.78;
>> Prof(2).Name='Fohl';
>> Prof(2).Groesse=1.82;
```

```
>> Prof(1)

ans =

        Name: 'Meisel'
    Groesse: 1.7800
```

- Cell Array

```
>> Zelle=cell(2,2);
>> Zelle{1,1}='Beispiel';
>> Zelle{1,2}=[1 2; 4 4];
>> Zelle{2,1}=true;
>> Zelle{2,2}=uint8(42);
```

```
>> Zelle{1,2}

ans =

         1         2
         4         4
```




6

Modellierung zeitkont. Systeme

am Beispiel von MatLab

6.1 Kurze Übersicht zu MatLab

6.2 Einführung in MatLab

6.3 Modellierung mit Simulink

6.4 Modellierung hybrider Systeme mit Stateflow



6.2.1 Vektoren

Erzeugen und Initialisieren

```
x1 = [1 2 3]; % Zeilenvektor (Trennung durch Leerzeichen)
x2 = [4, 5, 6]; % Zeilenvektor (Trennung durch Kommata)
```

```
x3 = [1; 2; 3]; % Spaltenvektor (Trennung durch Semikolon)
x4 = [4 5 6]'; % Spaltenvektor durch Transponieren eines
               % Zeilenvektors
```

```
x5 = 4:6; % Erzeugen und Initialisieren eines Vektors
          % [4 5 6]
x6 = 1:2:5; % Erzeugen und Initialisieren eines Vektors
            % [1 3 5] (von 1 bis 5 in Zweierschritten)
```

```
x8 = zeros(1, 3); % erzeugt Zeilenvektor [0 0 0]
x9 = zeros(3, 1); % erzeugt Spaltenvektor [0 0 0]'
x10 = ones(1, 3); % erzeugt Zeilenvektor [1 1 1]
x11 = rand(3,1); % erzeugt Spaltenvektor (3 Zeile, 1 Spalte)
                 % mit Zufallszahlen
```



Gegeben ist: $x7 = [1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8];$

Zugriff auf Elemente

```
% Der Index beginnt bei Matlab bei 1 (nicht bei 0, wie in c/Java)
a = x7(2);           % Zugriff auf das 2. Element (a=2)
b = x7(2:4);         % Zugriff auf die Elemente 2 bis 4
                     % d.h. : b = [2 3 4]
```



Gegeben ist:

$$\mathbf{x}_1 = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix};$$
$$\mathbf{x}_2 = \begin{bmatrix} 4 & 5 & 6 \end{bmatrix};$$
$$\mathbf{x}_3 = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix};$$

Einfache Rechenoperationen

```
c = 3*x1;           % Multiplikation : Skalar mit Vektor
                    % d.h. : c = [3 6 9]
```

```
d = x1*x2';         % Skalarprodukt zweier Zeilenvektoren
                    % x2 muss zuvor in Spaltenvektor umgewandelt werden
                    % d = 32
e = x1*x3;           % Skalarprodukt Zeilenvektor mit Spaltenvektor
                    % e = 14
```

```
f = x1.*x2;         % Elementweise Multiplikation zweier Vektoren
                    % d.h. : g = [4 10 18]
```

```
g = x1+x2;          % Addition zweier Vektoren
                    % d.h. : f = [5 7 9]
```



```
x1 = [1  2  3];  
x2 = [4, 5, 6];  
x3 = [1; 2; 3];
```

Wichtige Vektorfunktionen

```
h = [x1 x2];      % Konkatenieren zweiere Zeilenvektoren  
                  % d.h. : h = [1 2 3 4 5 6]  
  
j = norm(x1);     % Länge des Vektors (Euklidische Länge)
```



Übung : Vektoren in Matlab 1

```
x1 = [1 2 3];
```

```
x2 = [4, 5, 6];
```

```
x3 = [1; 2; 3];
```

Welche Matlab-Aufrufe sind falsch (→ Fehlermeldungen)?

a) `x1 + [2 3]`

b) `x1 + x3`

c) `x1 * x1`

d) `x3' * x1'`

e) `a = x1(0)`

f) `a = x2(x1(2))`



Übung : Vektoren in Matlab 2

$$\vec{a}_1 = (2, 5)$$

Geben Sie folgende Berechnungen in Matlab an.

$$\vec{b}_1 = \frac{\vec{a}_1}{|\vec{a}_1|}$$

$$\vec{b}_2 = (-a_{1,y}, a_{1,x})$$

Es ist zu prüfen, ob \vec{b}_1 und \vec{b}_2 senkrecht aufeinander stehen.

Es ist zu prüfen, ob \vec{b}_1 ein Einsvektor ist.



6.2.2 Matrizen

Erzeugen und Initialisieren

```
A1 = [1 2 3; 4 5 6];      % erzeugt 2x3-Matrix  
A2 = [1 2 ; 3 4 ; 5 6];   % erzeugt 3x2-Matrix  
A3 = A1';                 % A3 ist die Transponierte von A1
```

$$A_1 = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

$$A_2 = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}$$

$$A_3 = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$$

```
A4 = zeros(2, 2);         % erzeugt 2x2-Matrix mit Nullen  
A5 = ones(2, 2);          % erzeugt 2x2-Matrix mit Nullen  
A6 = eye(3,3);            % erzeugt 3x3-Einheitsmatrix
```

$$A_6 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$



Gegeben ist: $A_7 = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix};$

$$A_7 = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Zugriff auf Elemente

```
a = A7(2,3); % Zugriff auf das Element in der 2. Zeile
               % und der 3. Spalte , d.h. a = 6
b = A7(1,:);  % Zugriff auf alle Elemente der 1. Zeile
               % d.h. : b = [1 2 3]
c = A7(:,2);  % Zugriff auf alle Elemente der 2. Spalte
               % d.h. : c = [2 5 8]' (Spaltenvektor)
```

```
A8 = A7(2:3,1:2); %Zugriff auf eine Submatrix
                  % die Spalten 1 bis 2 in den Zeilen 2 bis 3
                  % d.h. : A8 = [4 5; 7 8]
```



Gegeben ist:

$$A_1 = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \quad A_9 = \begin{pmatrix} 1 & 2 & 1 \\ 1 & 0 & 2 \end{pmatrix}$$

Einfache Rechenoperationen

```
A10 = A1+A9;      % Addition zweier Matrizen
                  % Matrizen müssen gleiche Dimensionen haben !
A11 = 2*A9;       % Multiplikation einer Matrix mit einem Skalar
A12 = A1*A9';     % Matrizenmultiplikation
                  % Matrizen müssen verkettbar sein
A13 = A1.*A9;     % Elementweise Multiplikation
                  % Matrizen müssen gleiche Dimensionen haben !
```

$$A_{10} = \begin{pmatrix} 2 & 4 & 4 \\ 5 & 5 & 8 \end{pmatrix} \quad A_{11} = \begin{pmatrix} 2 & 4 & 2 \\ 2 & 0 & 4 \end{pmatrix} \quad A_{12} = \begin{pmatrix} 8 & 7 \\ 20 & 16 \end{pmatrix}$$
$$A_{13} = \begin{pmatrix} 1 & 4 & 3 \\ 4 & 0 & 12 \end{pmatrix}$$



$$A_{14} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \quad A_1 = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

weitere nützliche Matrixoperationen

```
A15 = inv(A14)      % Inverse der Matrix A14
                    % Matrix muss quadratisch sein!
A16 = det(A14)      % Determinante der Matrix A15
                    % Matrix muss quadratisch sein!
```

```
A17 = reshape(A1, 1, 6); % Erzeuge aus der 2x3 Matrix A1 eine
                        % 1x6-Matrix (A17 = [1 4 2 5 3 6])
                        % Anm.: Spaltenweise Umsortierung !
A18 = A1(end:-1:1 , :); % Zeilenreihenfolge umkehren
A19 = A1(: , end:-1:1); % Spaltenreihenfolge umkehren
```

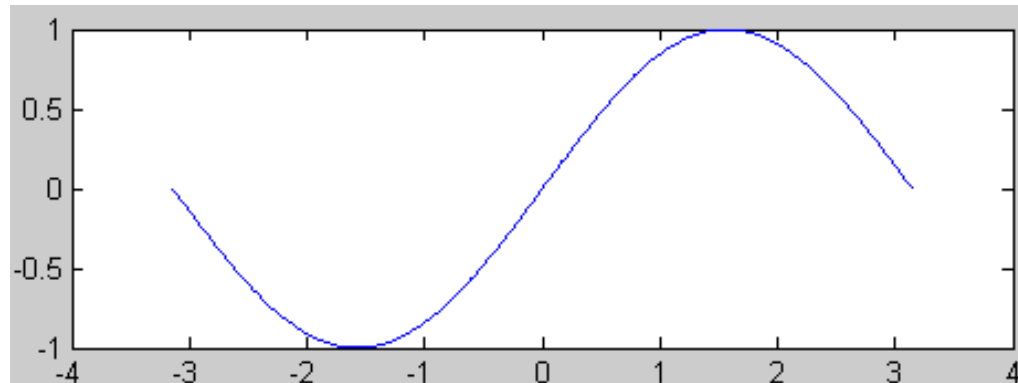
$$A_{18} = \begin{pmatrix} 4 & 5 & 6 \\ 1 & 2 & 3 \end{pmatrix} \quad A_{19} = \begin{pmatrix} 3 & 2 & 1 \\ 6 & 5 & 4 \end{pmatrix}$$



6.2.3 Grafische Ausgaben (*sehr kleine Auswahl*)

plot

```
x = -pi:0.01:pi;    % Vektor von -Pi bis + Pi in 0.01-er Schritten  
y = sin(x);         % Vektor der Sinuswerte  
plot(x,y);          % Funktion Plotten
```



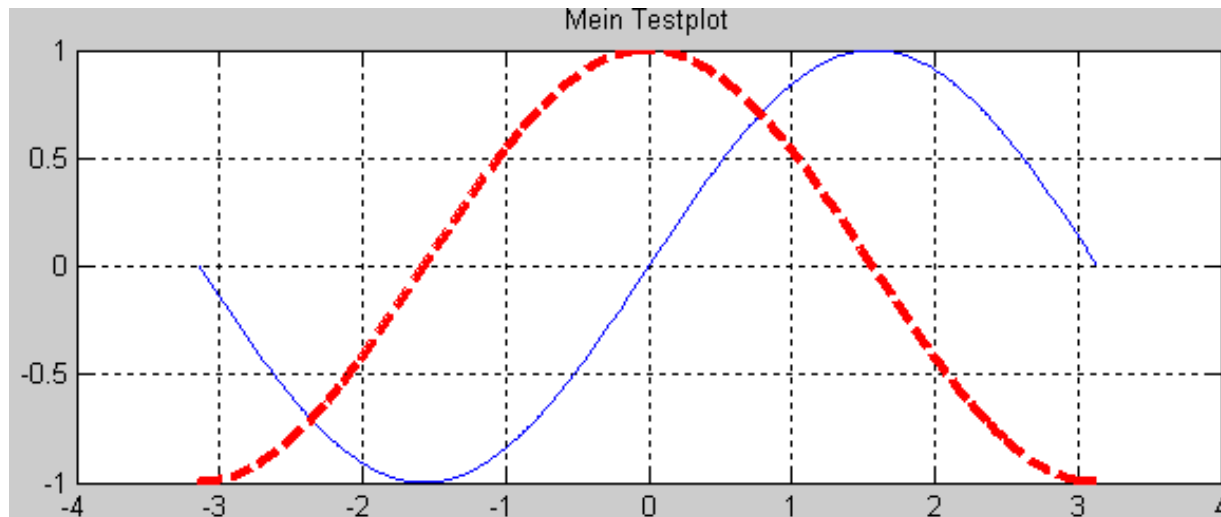


```
x = -pi:0.01:pi;    % Vektor von -Pi bis + Pi in 0.01-er Schritten
y1 = sin(x);        % Vektor der Sinuswerte
plot(x,y1);         % Funktion plotten

hold on;            % weitere Funktion in die gleiche "figure" plotten

y2 = cos(x);        % Vektor der Cosinuswerte
plot(x,y2, 'r--', 'linewidth', 3); % Funktion plotten (rot gestrichelt)
                                % Liniendicke = 3

title('Mein Testplot'); % Plot beschriften
grid on;            % Raster einblenden
```



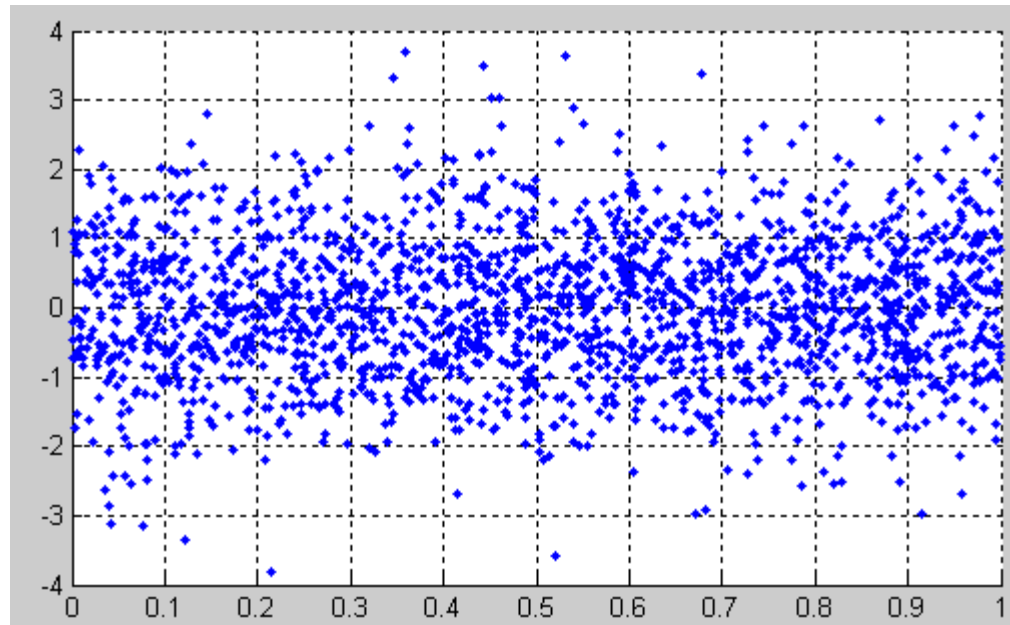


Symbol	Farbe	Symbol	Marker	Symbol	Linienart
b	blau	.	Punkt	-	durchgezogen
g	grün	o	Kreis	:	punktiert
r	rot	x	Kreuz	-.	strich-punktiert
c	cyan	+	Plus	--	strichliert
m	magenta	*	Stern		
y	gelb	s	Quadrat		
k	schwarz	d	Diamant		
w	weiß	v	Dreieck (unten)		
		^	Dreieck (oben)		
		<	Dreieck (links)		
		>	Dreieck (rechts)		
		p	Pentagramm		
		h	Hexagramm		



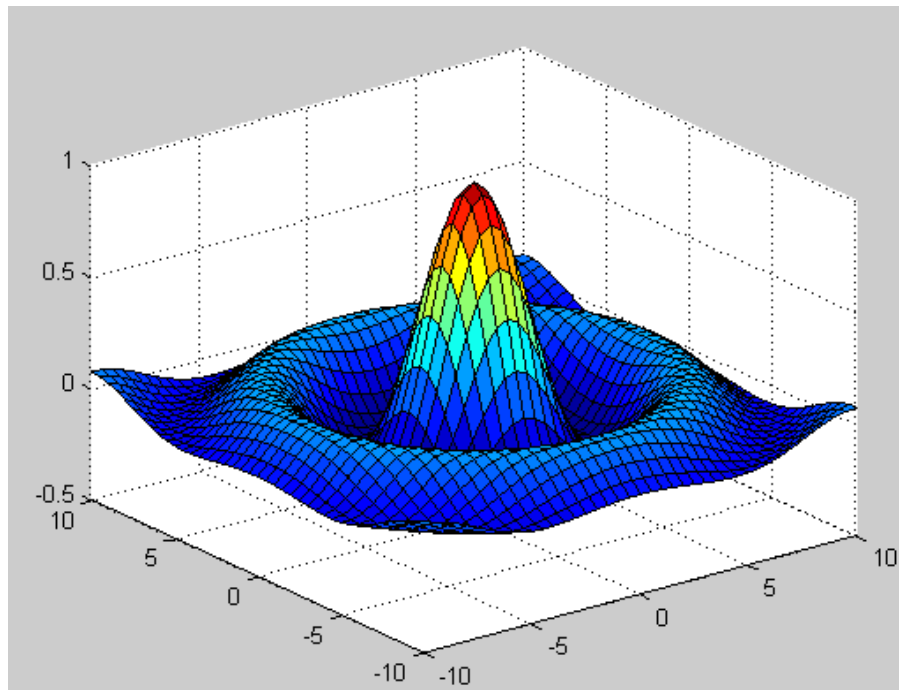
Scatterplot

```
x=rand(1,2000);           % Zeilenvektor mit 2000 gleichverteilten Werten
y=randn(1,2000);          % Zeilenvektor mit 2000 normalverteilten Werten
scatter(x,y,8,'filled' );  % Scatterplot mit gefüllten Punkten der Größe 8
grid on;                  % Raster einblenden
```



Surfaceplot

```
[x y] = meshgrid(-10 : 0.5 : 10); % erzeuge für jeden Gitterpunkt ein x-y-Paar
R      = sqrt(x.^2 + y.^2) + eps; % Berechne Abstand eines Gitterpunktes von (0,0).
                                     % x.^2: quadriere jeden x-Wert des Vektors x
                                     % Addiere einen sehr, sehr kl. Wert eps
                                     % --> damit wir R nie ganz 0.
Z      = sin(R) ./ R;               % Dividiere Zähler und Nenner elementweise
surf(x,y,Z);                       % Plotte
```



Anm.:

```
[x y] = meshgrid(-1:1:1);
```

erzeugt 2 Matrizen

```
x =
    -1     0     1
    -1     0     1
    -1     0     1
```

```
y =
    -1    -1    -1
     0     0     0
     1     1     1
```

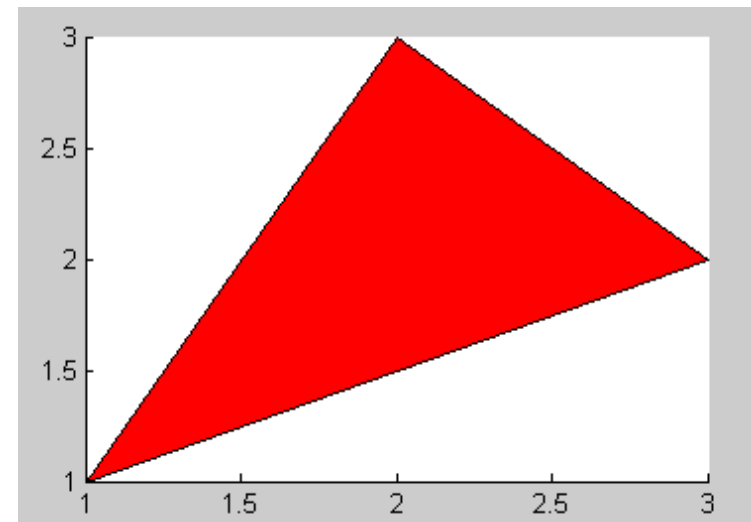

Patch

→ Zeichnen einfacher grafischer Objekte (Polygone).

```
Red=[1 0 0];  
Corners=[1 1; 2 3; 3 2; 1 1];  
patch(Corners(:,1)',Corners(:,2)',Red )
```

Zeilenvektor der
x-Koordinaten
= [1 2 3 1]

Zeilenvektor der
y-Koordinaten
= [1 3 2 1]



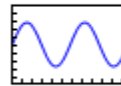


Übersicht: Plot-Funktionen

nicht vollständig

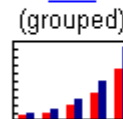
**Line
Graphs**

[plot](#)



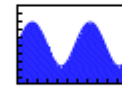
**Bar
Graphs**

[bar](#)



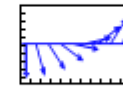
Area Graphs

[area](#)



**Direction
Graphs**

[feather](#)



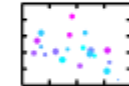
**Radial
Graphs**

[polar](#)

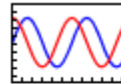


**Scatter
Graphs**

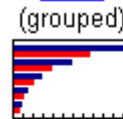
[scatter](#)



[plotyy](#)



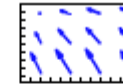
[barh](#)



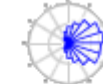
[pie](#)



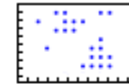
[quiver](#)



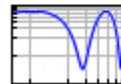
[rose](#)



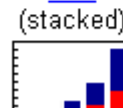
[spy](#)



[loglog](#)



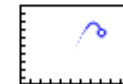
[bar](#)



[fill](#)



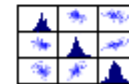
[comet](#)



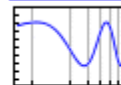
[compass](#)



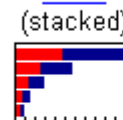
[plotmatrix](#)



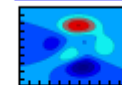
[semilogx](#)



[barh](#)



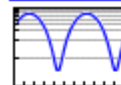
[contourf](#)



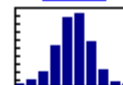
[ezpolar](#)



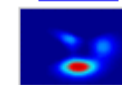
[semilogy](#)



[hist](#)



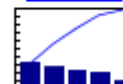
[image](#)



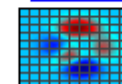
[stairs](#)

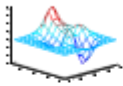
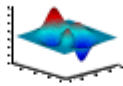
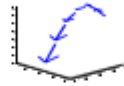
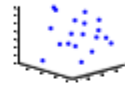
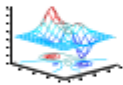
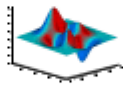
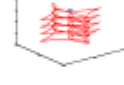
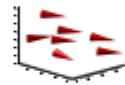
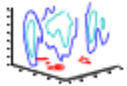
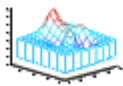
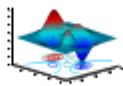
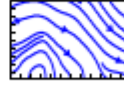
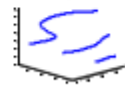
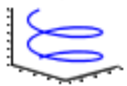
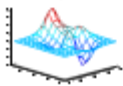
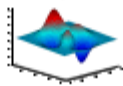
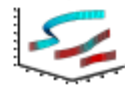
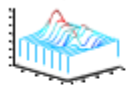
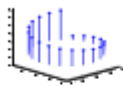
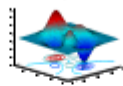
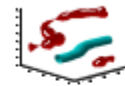
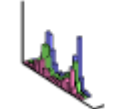


[pareto](#)



[pcolor](#)



**Line Graphs****Mesh
Graphs
and Bar
Graphs****Area
Graphs and
Constructive
Objects****Surface
Graphs****Direction
Graphs****Volumetric
Graphs**[plot3](#)[mesh](#)[pie3](#)[surf](#)[quiver3](#)[scatter3](#)[contour3](#)[meshc](#)[fill3](#)[surf1](#)[comet3](#)[coneplot](#)[contourslice](#)[meshz](#)[patch](#)[surfz](#)[streamslice](#)[streamline](#)[ezplot3](#)[ezmesh](#)[cylinder](#)[ezsurf](#)[streamribbon](#)[waterfall](#)[stem3](#)[ellipsoid](#)[ezsurfz](#)[streamtube](#)[bar3](#)[sphere](#)



6.2.4 Eigene Funktionen

```
function [out1, out2, ...] = funktionsname (in1, in2, .....)
```

```
% Sinus für Winkel in Grad  
function y = singrad(x)  
    y = sin(x*pi/180);  
return
```

Wichtig: als singrad.m abspeichern

Aufruf mit Skalar

```
>> singrad(45)  
ans =  
  
    0.7071
```

Aufruf mit Vektor

```
singrad([15 30 45])  
ans =  
  
    0.2588    0.5000    0.7071
```



Lokale Variablen und lokale Funktionen

```
% Sinus für Winkel in Grad
function y = singrad2(x)
    WinkelInBogenmass = x*pi/180;
    y = sin(WinkelInBogenmass);
return
```

Lokale Variablen müssen nicht deklariert werden. Sie werden beim Verlassen der Funktion gelöscht.

```
% Sinus für Winkel in Grad
function y = singrad3(x)
    y = sin(GradInBogenmass(x));
return

% lokale Funktion
function bog = GradInBogenmass(grd)
    bog = grd*pi/180;
return
```

Lokale Funktionen sind nur innerhalb des m-Files (hier `singrad3.m`) bekannt.



Globale Variablen

```
% Globale Variable Inc hochzählen
function AddiereZuInc(x)
    global Inc;          % Glob. Variable deklarieren

    if isempty(Inc) % falls nicht existiert -> Meldung
        error('Globale Variable Inc existiert nicht.');
```

end

```
    Inc = Inc + x;      % Inc hochzählen
    return
```

Aufruf

```
>> global Inc
>> Inc=0;
>> AddiereZuInc(3);
>> Inc

Inc =

    3
```

Globale Variablen müssen vor Aufruf der Funktion mit `global variable` angelegt werden.

Globale Variablen werden mit `clear global` gelöscht.



Statische (persistente) Variablen

```
% Akkumuliert die übergebenen Werte
function y = Akkumuliere(x)
    persistent Akku;    % Stat. Variable anl.

    if isempty(Akku)    % bei Erstverwendung
        Akku = 0;       % initialisieren
    end

    Akku = Akku + x;    % Akku hochzählen
    y     = Akku;       % Akku ausgeben
return
```

Stat. Variablen bleiben zwischen den Aufrufen erhalten

werden gelöscht durch

- `clear functions`
- Änderung des m-Files

und sind nur innerhalb der Funktion sichtbar.

mehrmaliger Aufruf

```
>> Akkumuliere(2);
>> Akkumuliere(3);
>> Akkumuliere(1)
ans =

     6
```



6.2.5 Ablaufsteuerung

Schleifensteuerung mit for end

```
% Berechnet x1=0+5+10+15+20
x1=0;
- for i=0:5:20
    x1 = x1+i;
end
```

```
% Berechnet x2=0+5+10+15+20
x2=0;
- for i=[5 10 15 20]
    x2 = x2+i;
end
```

Schleifensteuerung mit while end

```
% Berechnet x2=0+5+10+15+20
x3=0; i=0;
- while i<=20
    x3=x3+i;
    i=i+5;
end
```




Verzweigungen mit if elseif else end

```
x4 = 10;
resetAllowed = true;    % log. Variable anlegen und init.
cmd = 1;                % init
if (cmd == 1) && (resetAllowed == true)
    x4 = 0;
elseif cmd == 2    % keine bis beliebig viele elseif möglich
    x4 = x4 - 1;
else              % kein oder ein else möglich
    error('kann nicht sein');
end
```



Verzweigungen mit switch ... case

```
a=10;  
flag = 'update';  
switch flag  
    case 'init'  
        a=0;  
    case {'update', 'inc'}  
        a=a+1;  
    otherwise  
        error('kann nicht sein');  
end
```



6

Modellierung zeitkont. Systeme

am Beispiel von MatLab

6.1 Kurze Übersicht zu MatLab

6.2 Einführung in MatLab

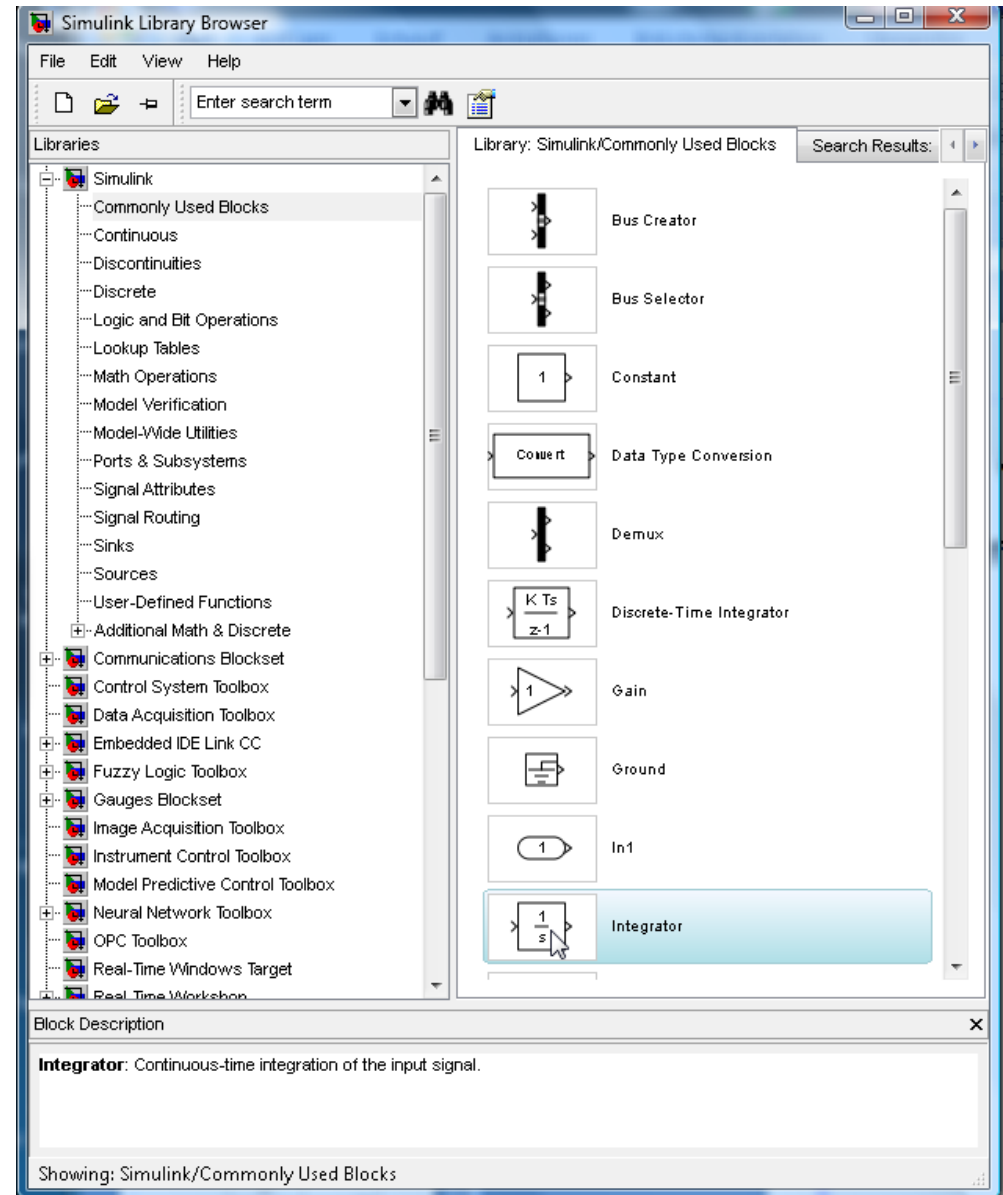
6.3 Modellierung mit Simulink

6.4 Modellierung hybrider Systeme mit Stateflow



6.3.1 Simulink – Übersicht

Start mit `>> simulink`





6.3.2 Simulink – Vor-und Nachteile

Vorteile:

- grafische, datenflußorientierte Modellierung
- hierarchische Modelle
- Zugriff auf den vollen Matlab-Funktionsumfang
- sehr umfangreiche grafische Ausgabemöglichkeiten
- gemischte diskrete/kontinuierliche Systeme

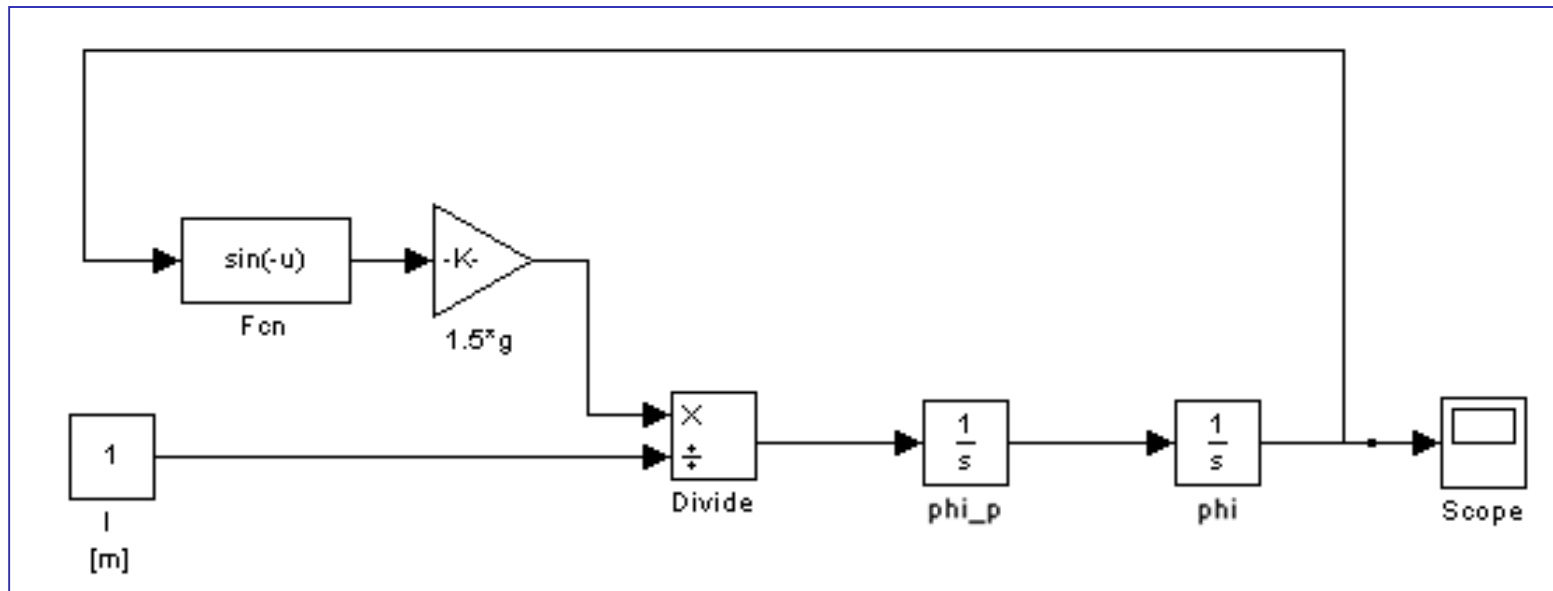
Nachteile:

- Systeme mit vielen Ereignissen werden leicht unübersichtlich (→ besser Stateflow)
- keine repetitiven Strukturen (→ besser Stateflow)



6.3.3 Blockorientierte Modellierung

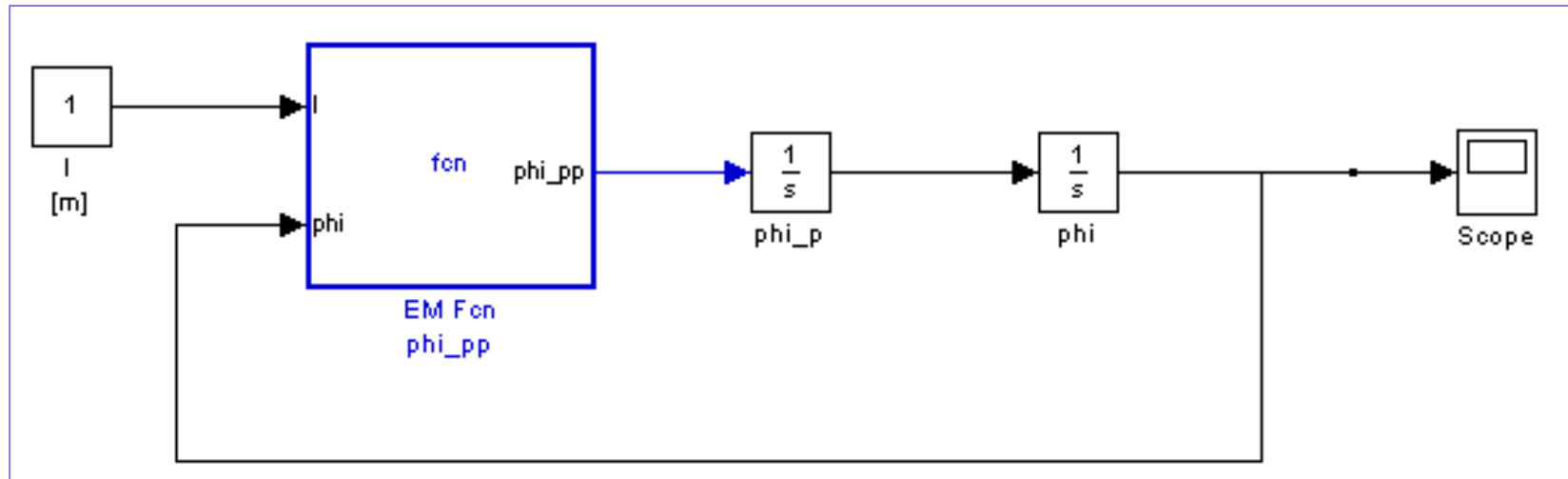
Was für eine Differentialgleichung ist hier modelliert?





6.3.4 Embedded-Matlab-Functions (bzw. Matlab-Functions)

Häufig ist es einfacher und übersichtlicher, mathematische Verknüpfungen nicht blockorientiert zu realisieren, sondern durch eine „*Embedded Matlab Function*“



```
function phi_pp = fcn(l, phi)
% Embedded MATLAB

phi_pp = 3/2*9.81*sin(-phi)/l;
```

Anm.: je nach MatLab-Version
nur „*Matlab-Function*“



zu **Embedded-Matlab-Funktionen (EM)**

EM stellt eine Untermenge der Matlab-Funktionen zur Verfügung.

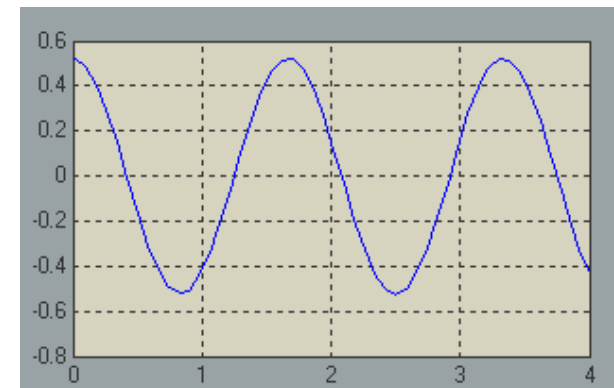
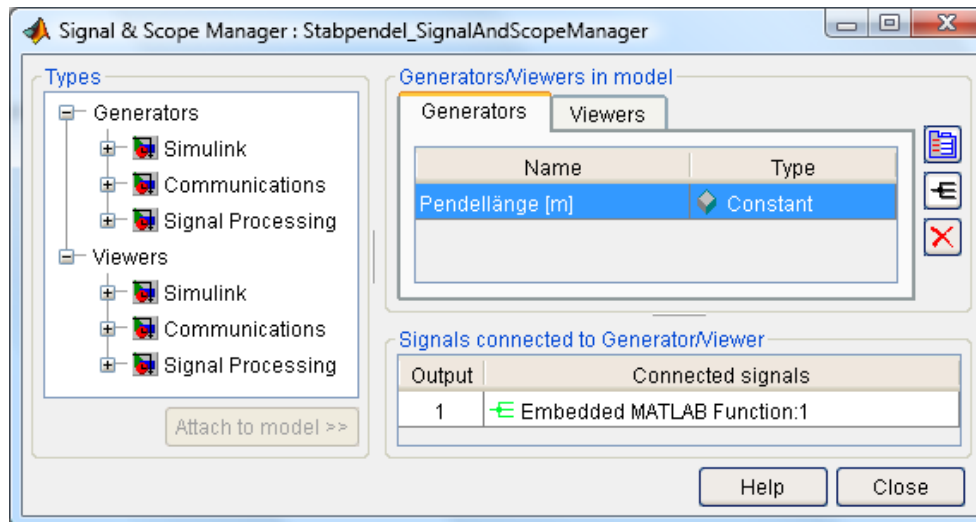
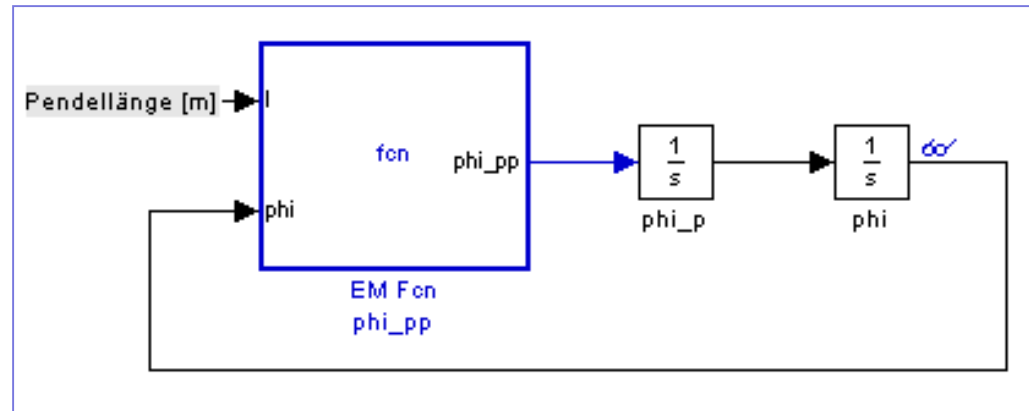
EM-Funktionen sind dann sinnvoll, wenn eine textuelle Beschreibung in der Matlab-Syntax einfacher ist als eine grafische Darstellung mit Simulink.

Aus **EM**-Funktionen lässt sich C-Code generieren, der auch auf externen eingebetteten System gestartet werden kann (→ *Matlab Realtime Workshop*).

Mit `eml.extrinsic('.....')` können auch non-**EM**-Funktionen im **EM**-Funktionsblock verwendet werden. Die Einbettbarkeit des Codes geht dann aber verloren.

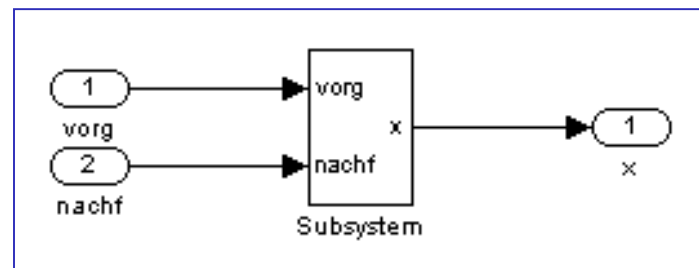
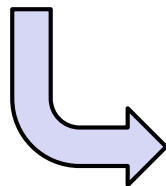
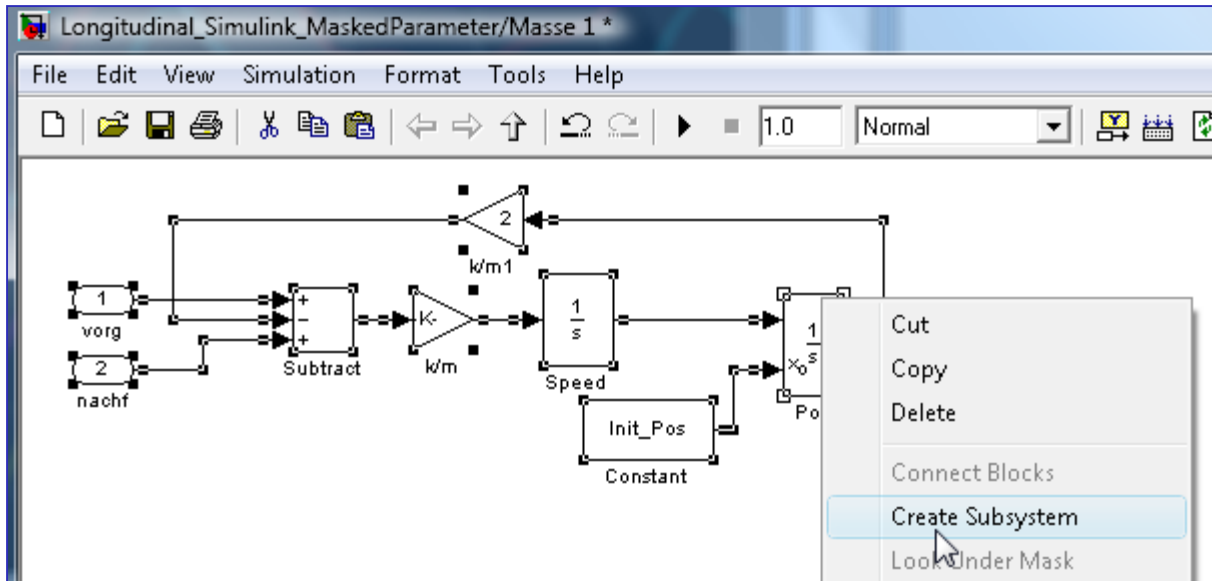
6.3.5 Signal & Scope Manager

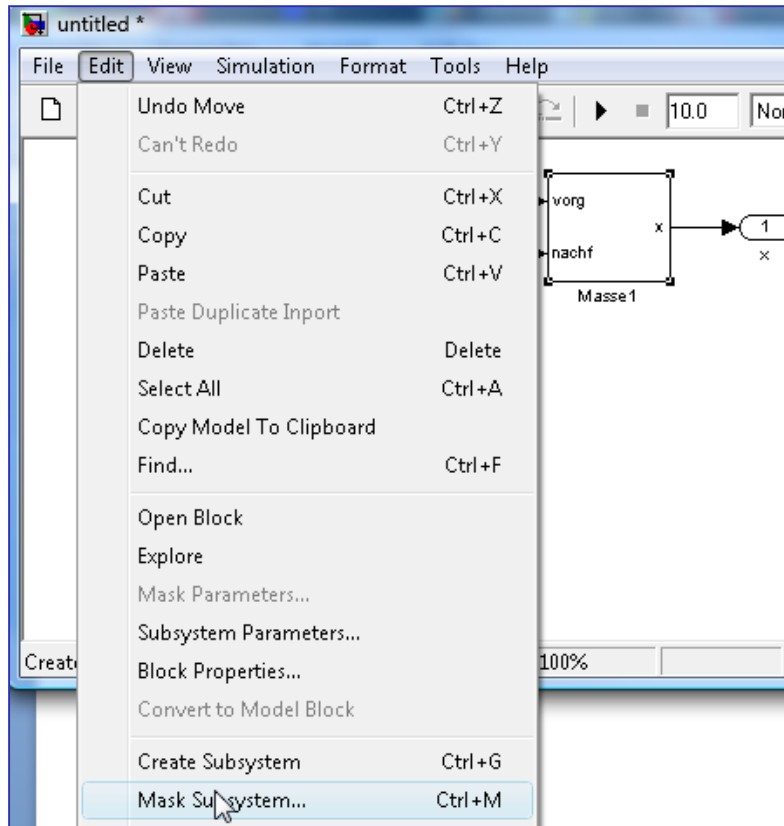
Die Modelle werden übersichtlicher, wenn alle Sources und Sinks mit Hilfe des Signal&Scope-Managers mit dem Modell verbunden werden.



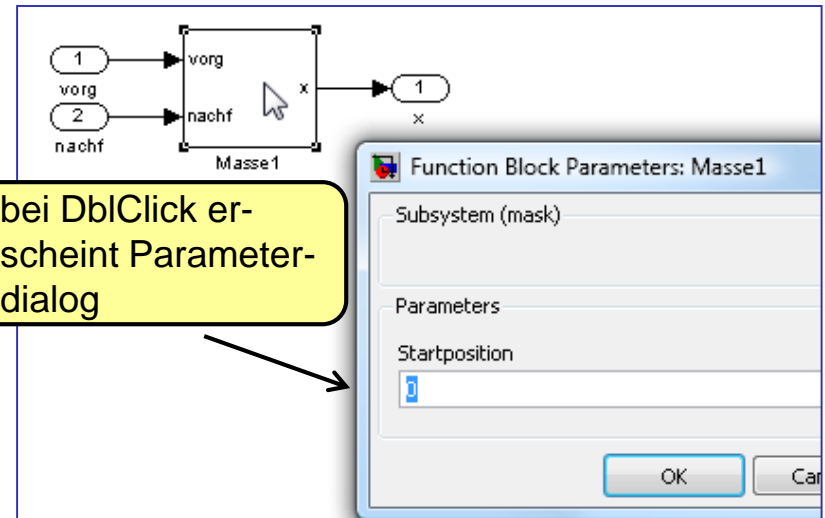
6.3.6 Subsysteme und Simulationsparameter

Durch Erzeugen von Subsystemen lassen sich Modelle hierarchisieren und leicht vervielfältigen.

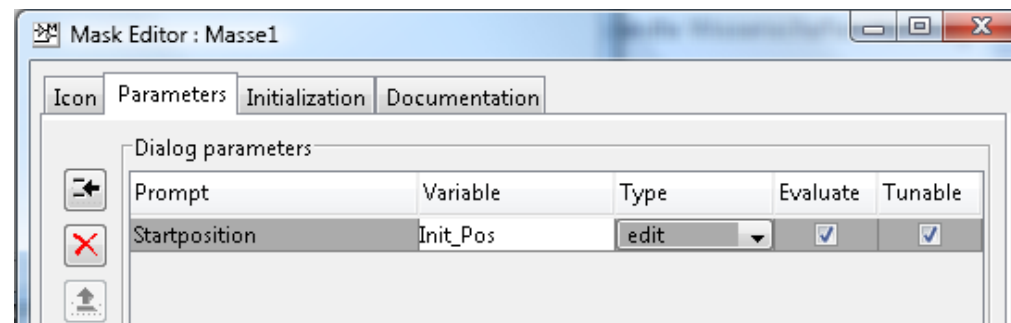
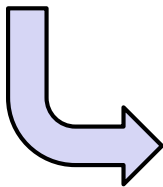


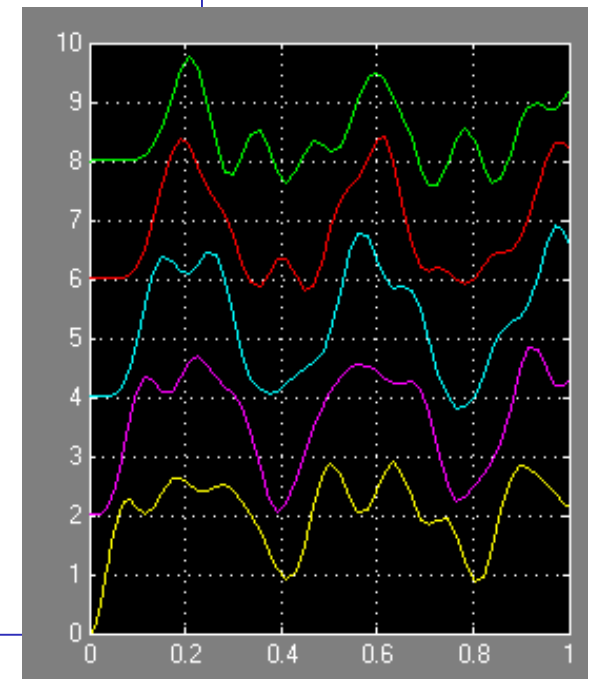
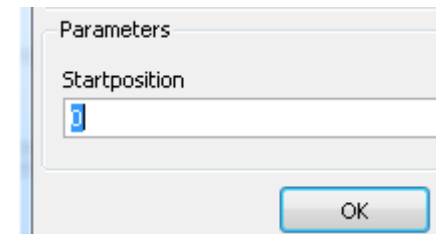
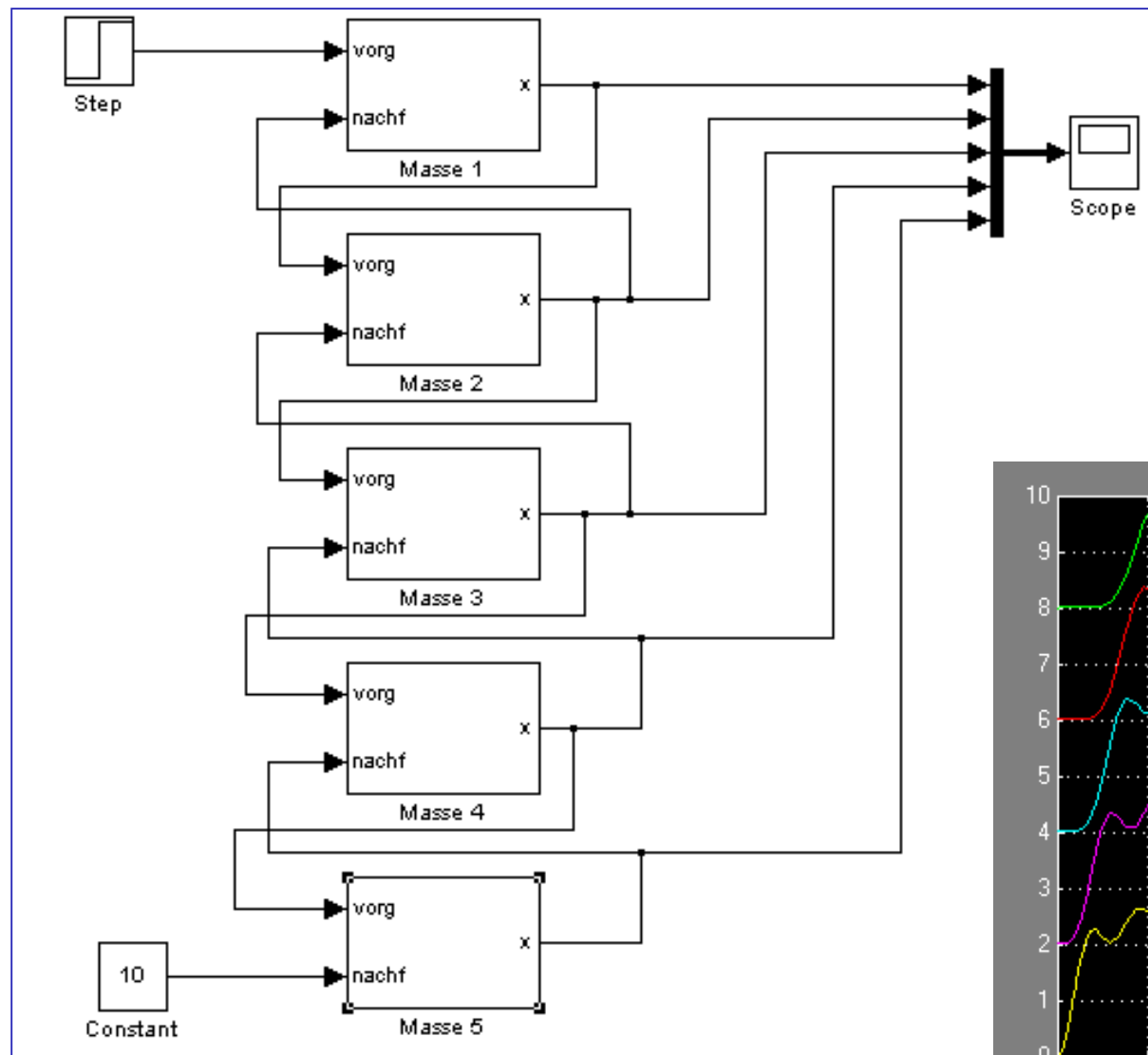


bei DbClick er-
scheint Parameter-
dialog

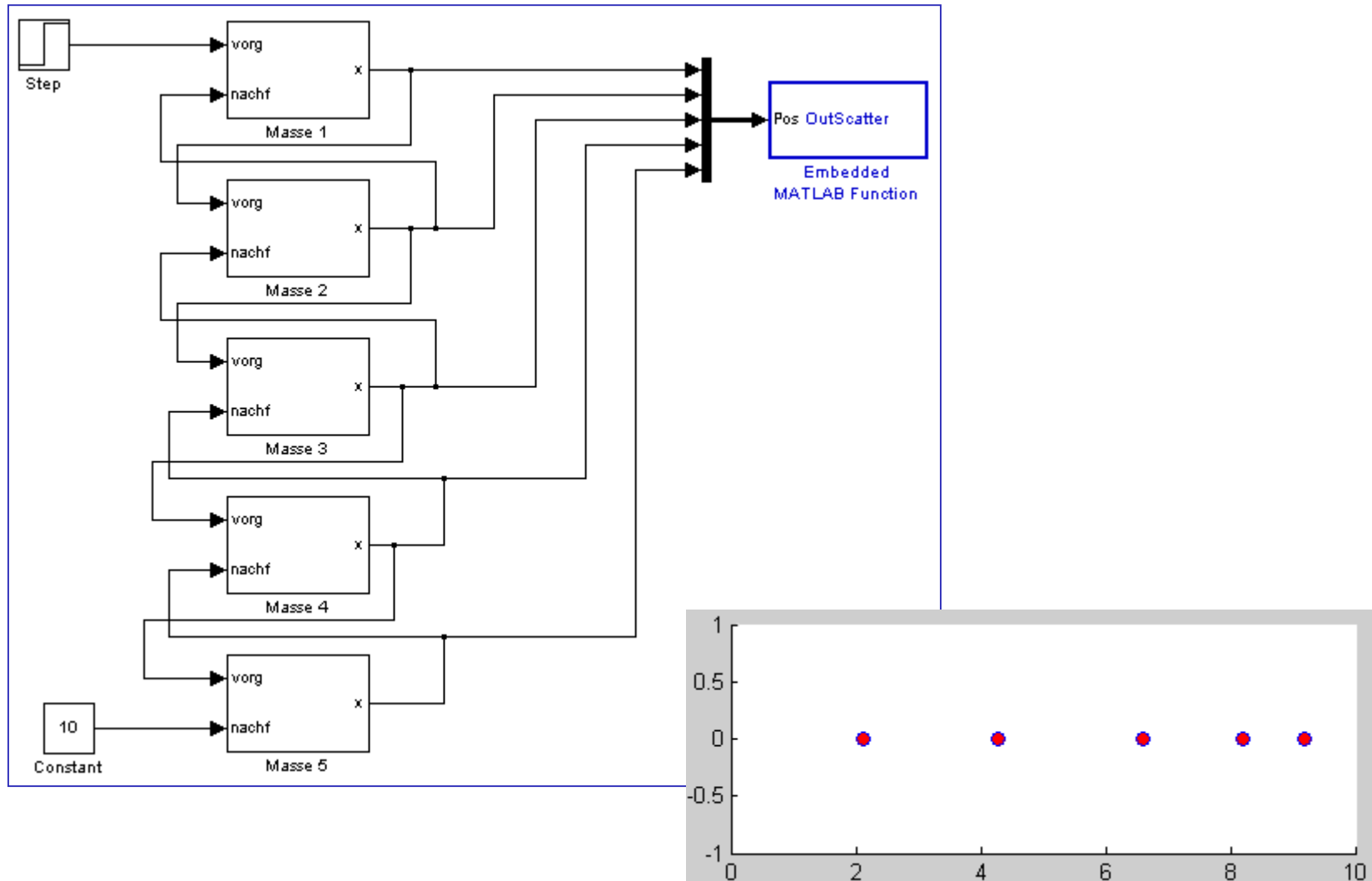


Konstanten eines Subsystems
können auf Wunsch im Parameter-
dialog des Subsystems angegeben
werden.





6.3.7 Matlab-Grafikfunktionen in Simulink nutzen





```
function OutScatter(Pos)
% Embedded MATLAB

eml.extrinsic('set', 'drawnow', 'scatter', 'gca');
persistent y 10;

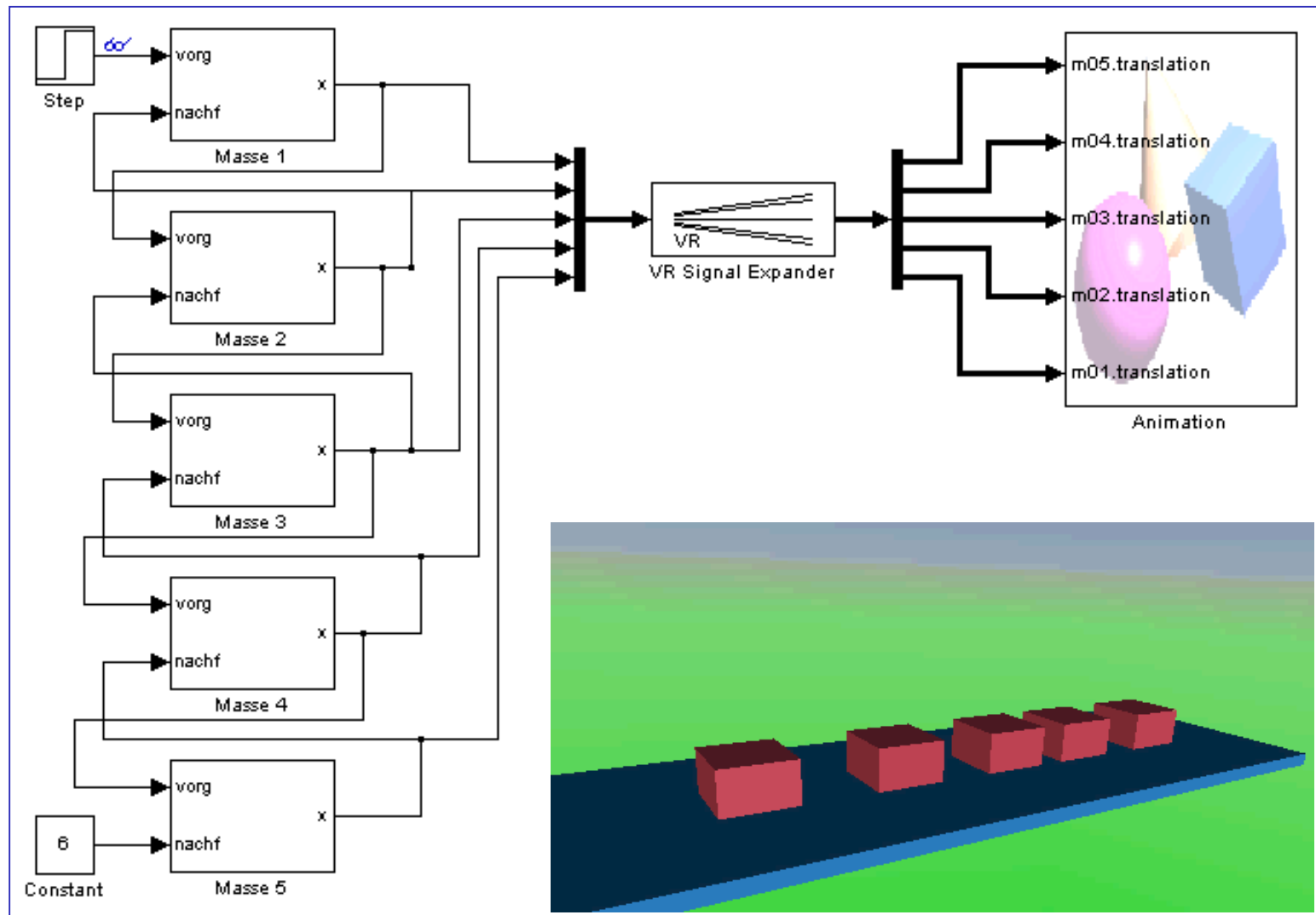
% Init-Parameter
k = 5;           % Anzahl der Massen
10 = 2;          % Abstand zwischen den Massen

% y-Werte anlegen und mit 0 initialisieren
y=zeros(1,k);

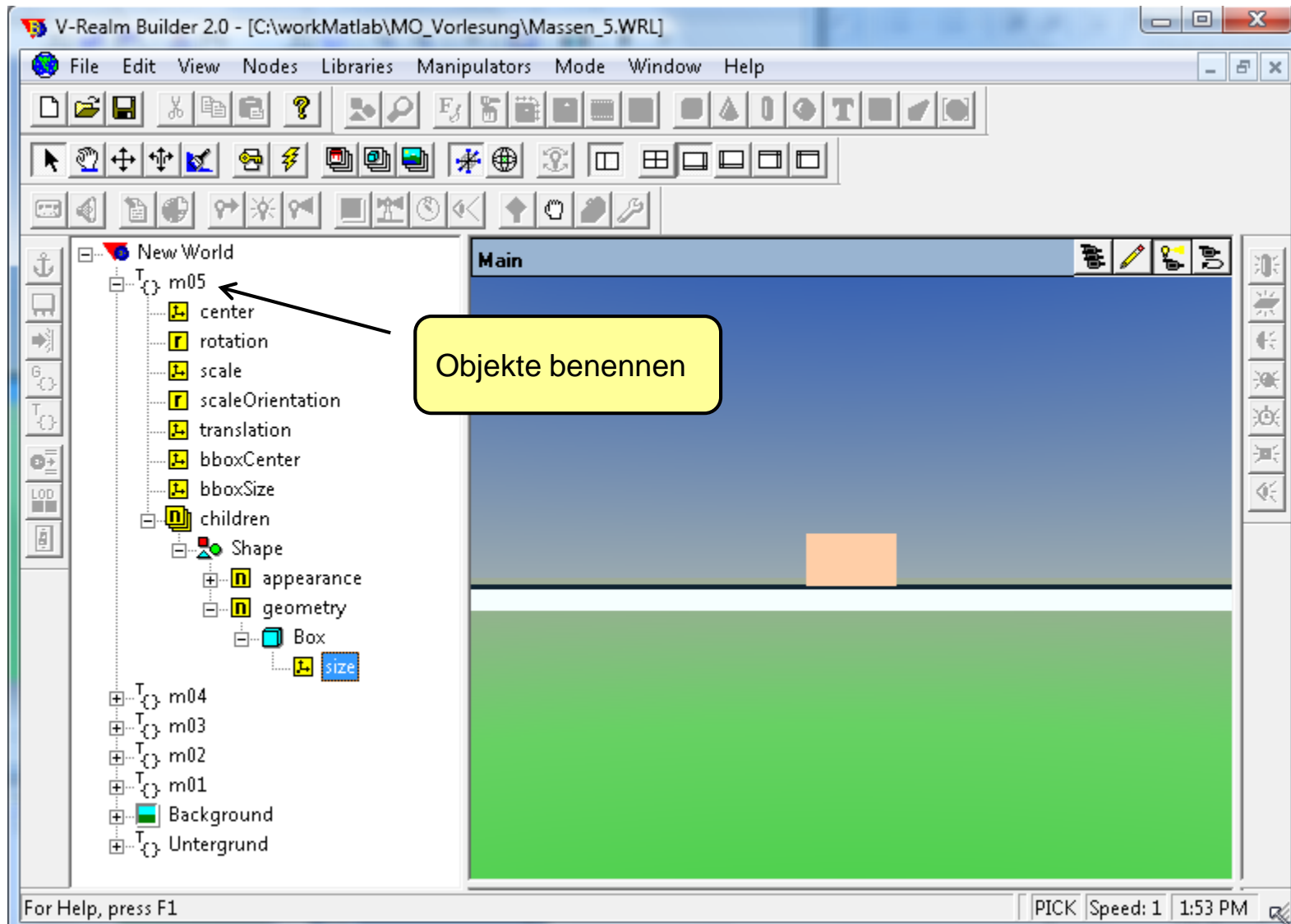
% Scatterplot zeichnen
scatter(Pos', y, 'o', 'MarkerFaceColor', [1 0 0]);
set (gca,'XLim', [0 k*10] ); % x-Bereich festhalten
drawnow;                    % jetzt zeichnen
end
```

Autoscale ausschalten

6.3.8 Virtual-Reality in Simulink nutzen



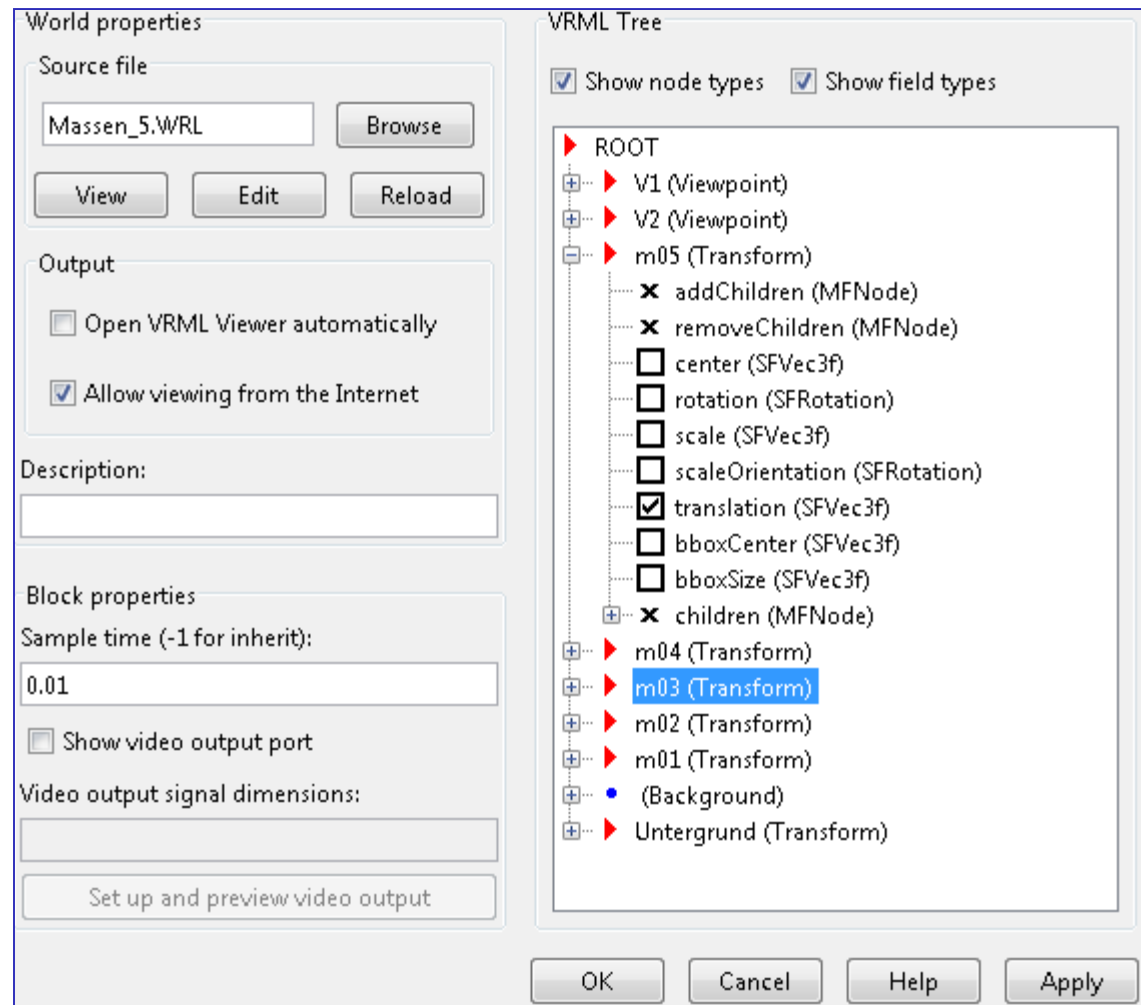
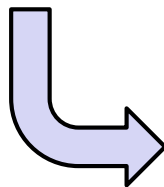
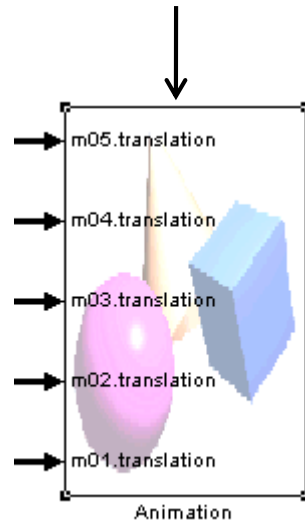
a) Graf. Objekte im V-Realm-Builders anlegen und abspeichern (.WRL)



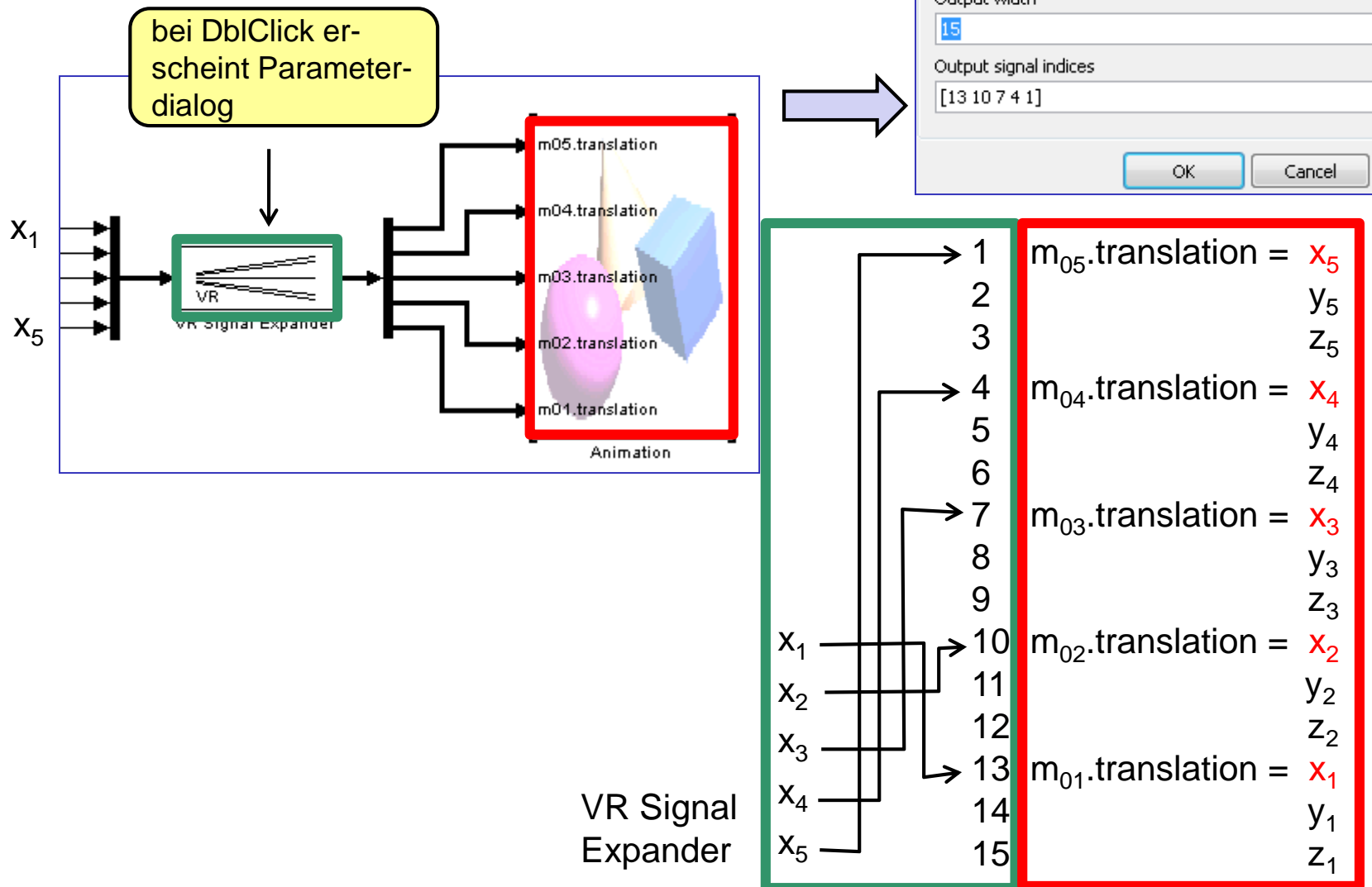


b) VRML-Modell laden und variable Parameter anwählen

bei DbIClick er-
scheint Parameter-
dialog



c) Animation mit Simulationsmodell verbinden





6

Modellierung zeitkont. Systeme

am Beispiel von MatLab

6.1 Kurze Übersicht zu MatLab

6.2 Einführung in MatLab

6.3 Modellierung mit Simulink

6.4 Modellierung hybrider Systeme mit Stateflow



6.4.1 Motivation

In vielen realen Systemen verändert sich das System durch bestimmte Ereignisse (zeit- oder zustandsabhängig). → *diskrete Ereignisse*

- Zustandsgrößen (Integrierer) verändern in bestimmten Situationen ihren Wert.
Beispiel: hüpfender Ball → Ballgeschwindigkeit ändert Wert und Vorzeichen.
- Systemparameter ändern sich in bestimmten Situationen.
Beispiel: Bei einer bestimmten Drehzahl schaltet das Automatikgetriebe in einen anderen Gang → anderes Übersetzungsverhältnis.
- Die Systemstruktur ändert sich in bestimmten Situationen.
s. Beispiel auf der nächsten Seite

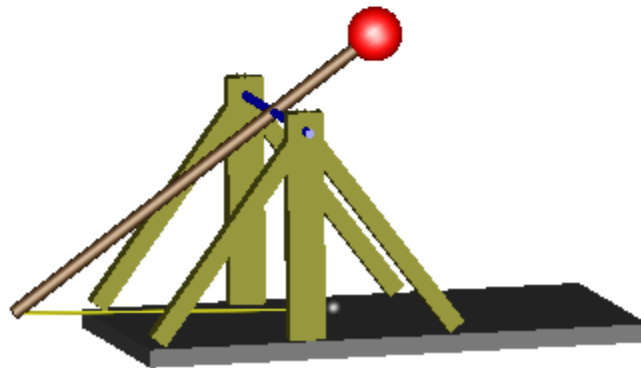
Bei der Modellierung von Systemen mit zeitkontinuierlichen und diskreten Anteilen spricht man von "**hybrider Modellierung**".

Hybride Modelle lassen sich besonders übersichtlich mit Stateflow modellieren.

Beispiel eines hybriden Modells mit veränderlicher Systemstruktur

Das Modell hat 3 Phasen:

1. Rutschphase: Zunächst sind Schleuderarm und Stein ein gemeinsames System. Der Stein rutscht über das Grundbrett (Gleitreibung).
2. Schleuderphase : Stein und Schleuderarm bilden immer noch ein Sytem. (Gravitation und Strömungswiderstand)
3. Wurfphase: Arm und Stein bewegen sich unabhängig voneinander.





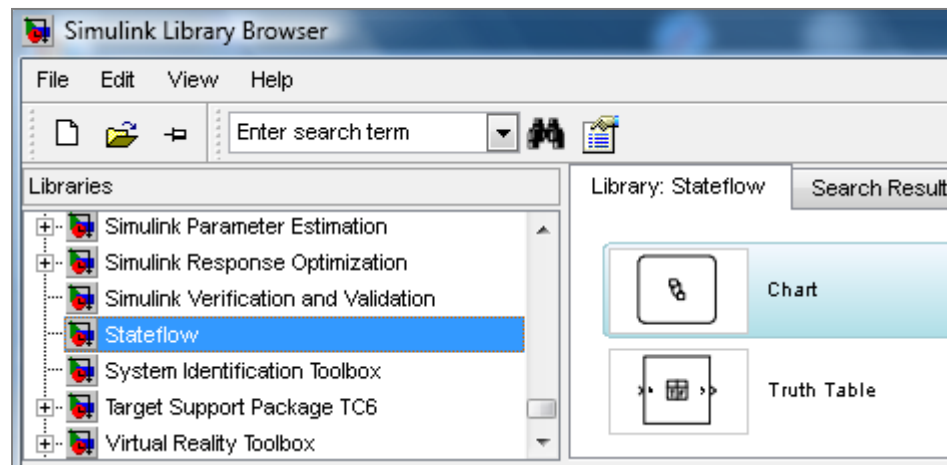
6.4.2 Übersicht zu Stateflow

6.4.2.1 Grundlegendes

Stateflow ist eine Erweiterung von Simulink zur Modellierung

- ereignisgesteuerter reaktiver Systeme und
- hybrider Systeme

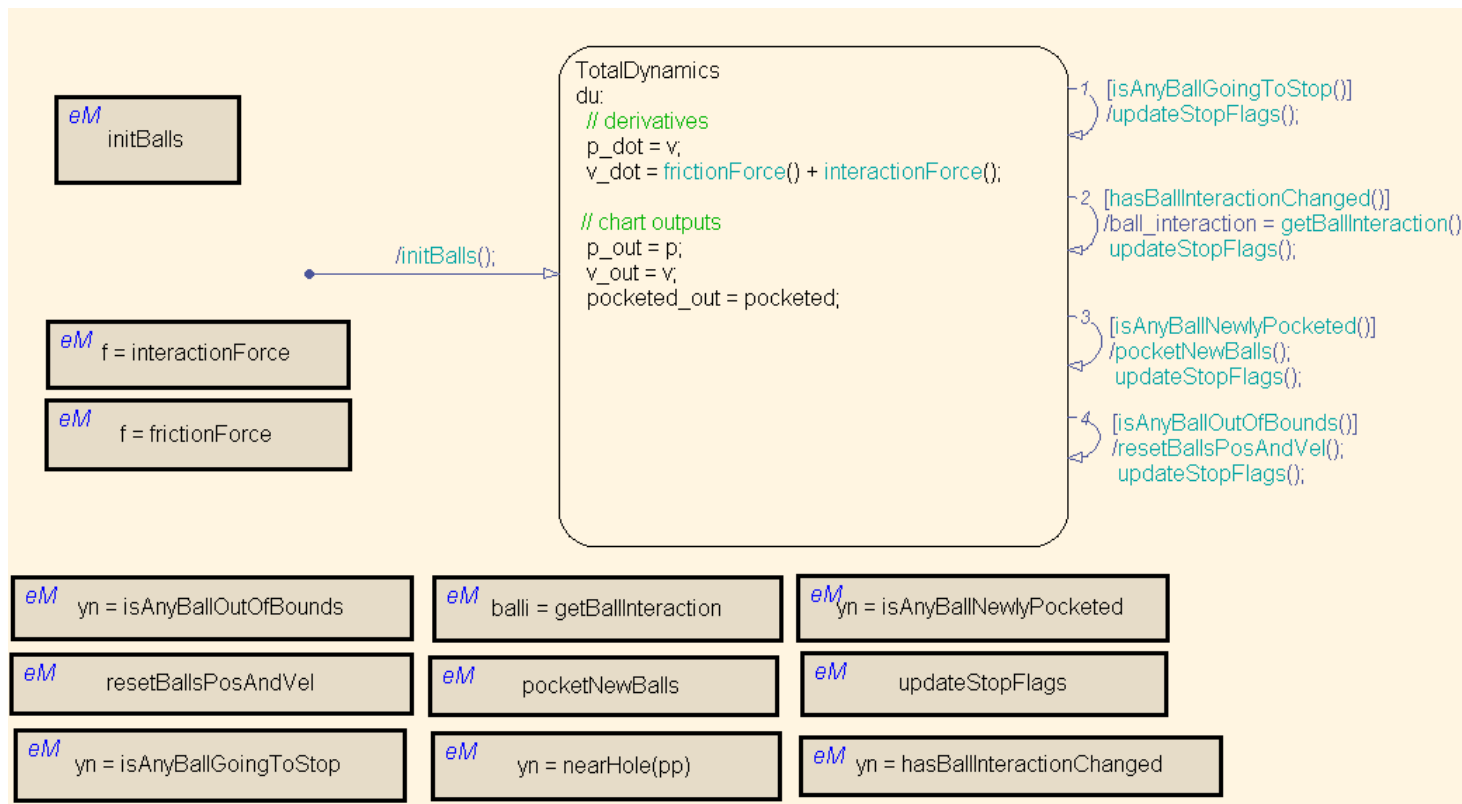
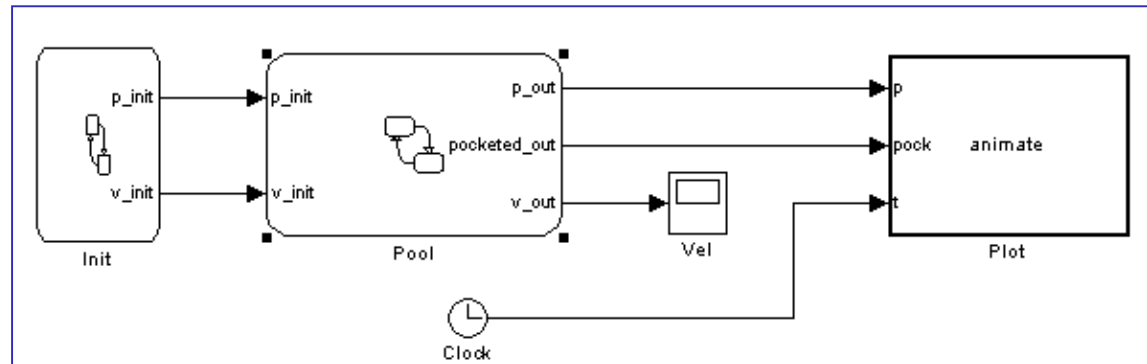
mit Hilfe von Zustandsautomaten.



Anm.: In der folgenden Darstellung wird nur ein kleiner Teil der Stateflow-Möglichkeiten dargestellt.

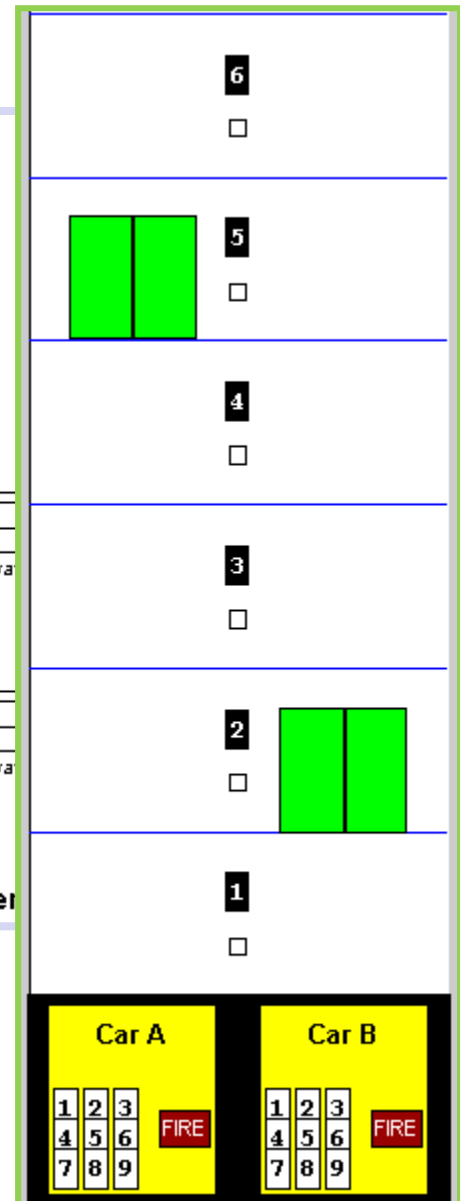
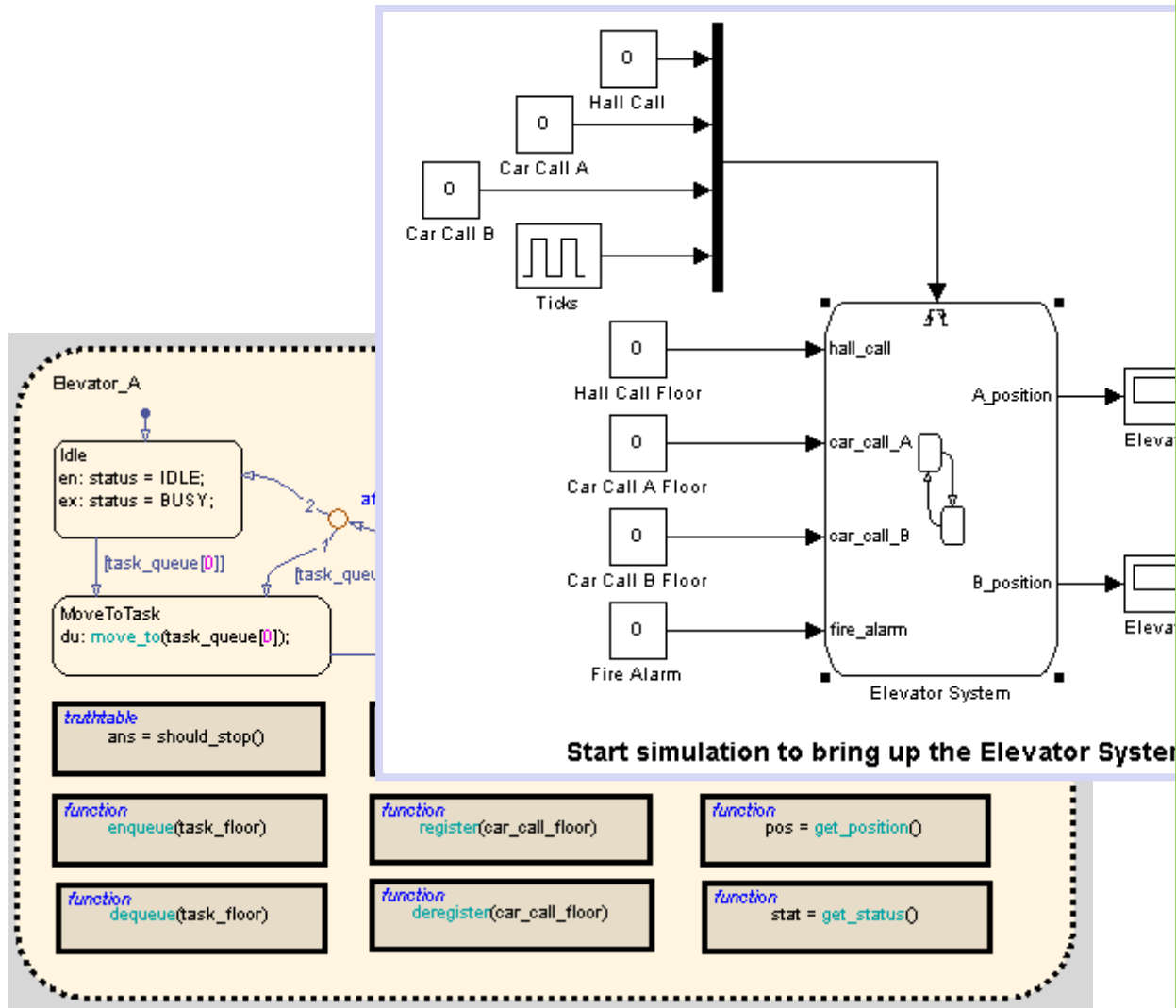


Beispiel: Billard



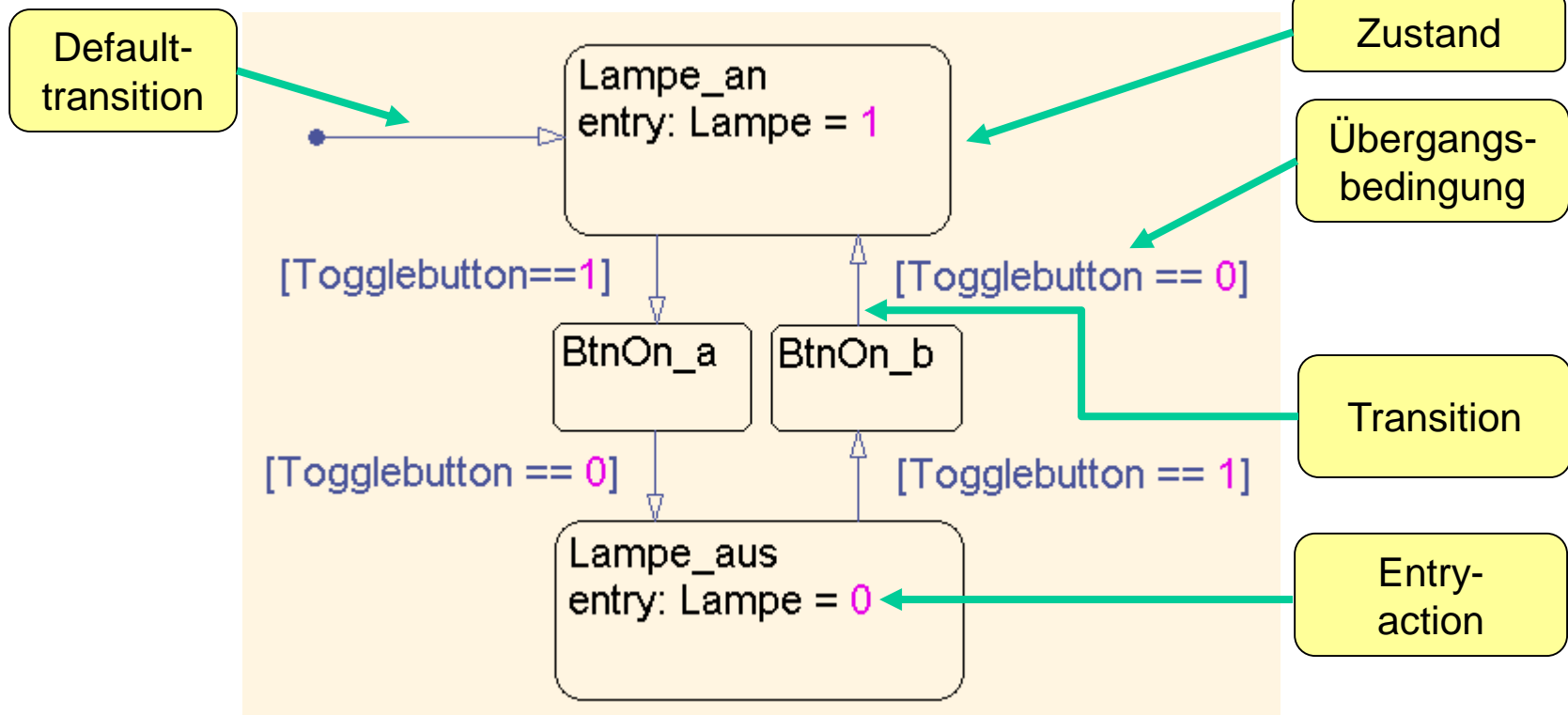
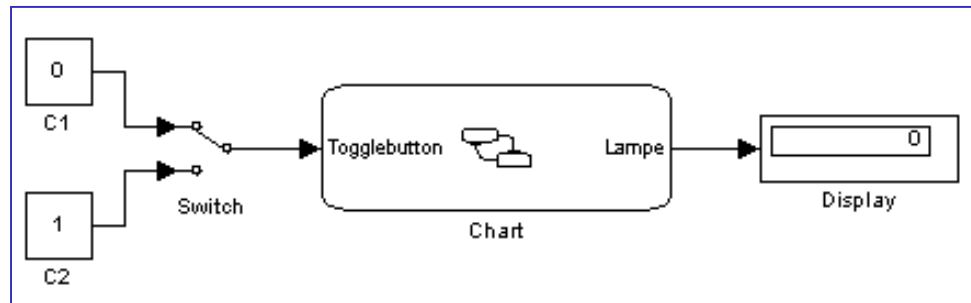


Beispiel: Fahrstuhlsteuerung



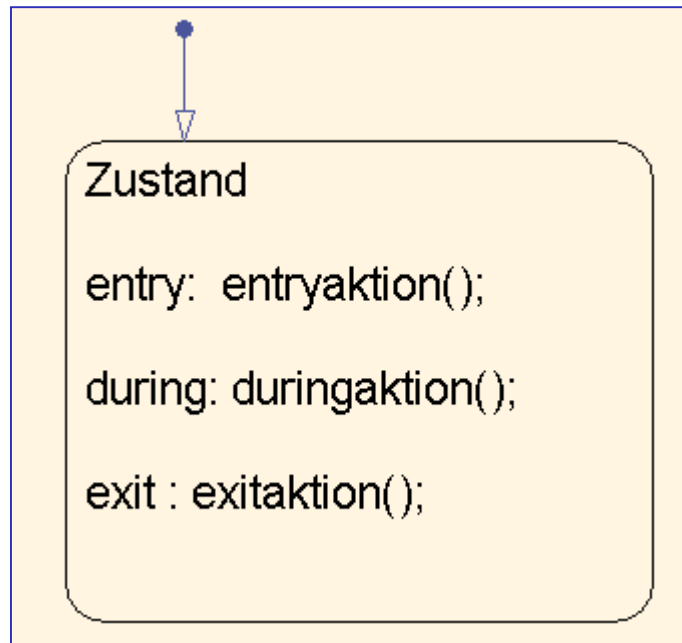


6.4.2.2 Grafische Elemente eines Charts an einem Beispiel





6.4.2.3 Zustände und Aktionsarten



entry action:

Wird beim Eintreten in den Zustand ausgeführt.

during action:

Wird ausgeführt, solange der Zustand aktiv ist und ist abhängig von der *Triggermethode* des Charts.

Triggermethode: **discrete**

→ von eingestellter SampleRate abhängig

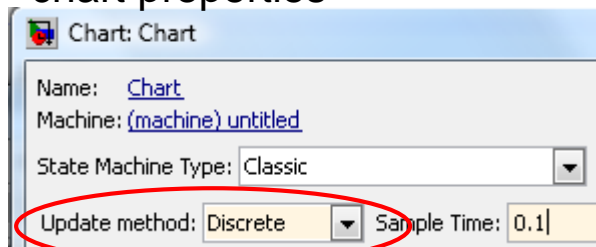
Triggermethode: **continuous**

→ bei jedem Integrationsschritt

Triggermethode: **inherited**

→ Takt wird von Simulink übernommen,
z.B. über äußeren „Trigger Event“

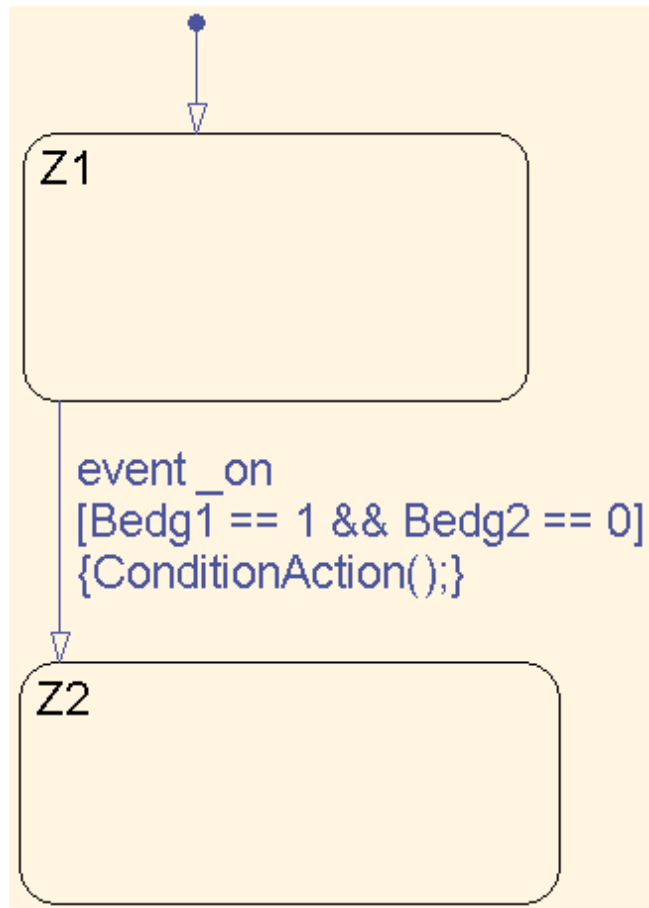
chart properties



Triggermethode



6.4.2.4 Transitionen



Event:

Transition kann nur gültig werden, wenn das Event auftritt. Wird das Event weggelassen, dann wird die Transition nur durch die Bedingung ausgelöst.

Bedingung: [....]

Die Transition wird nur ausgeführt, wenn die Bedingung den booleschen Wert *true* liefert. Wird die Bedingung weggelassen, wird *true* angenommen.

Bedingungsaktion: { }

Wird noch vor der Transition ausgeführt.

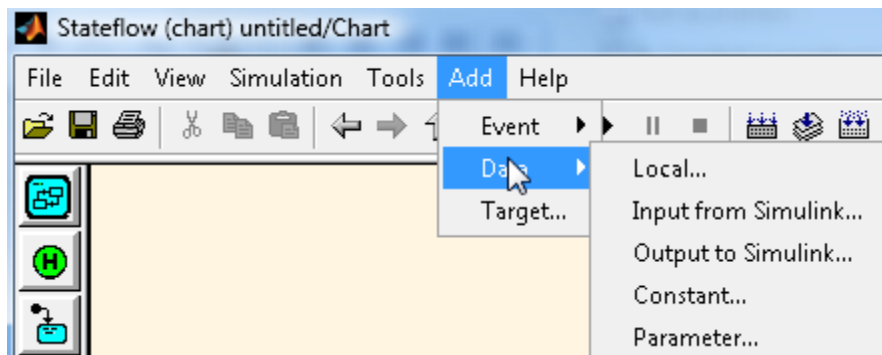


6.4.2.5 Datenarten von Stateflow

Die in der Chart benötigten Daten werden mit „*Add → Data → ...*“ angelegt. Hierbei ist zu unterscheiden zwischen:

- a. Zustandsgrößen (Local) → continuous/discrete
- b. Eingangsgrößen (Input from Simulink)
- c. Ausgangsgrößen (Output to Simulink)
- d. Konstanten
- e. Parameter

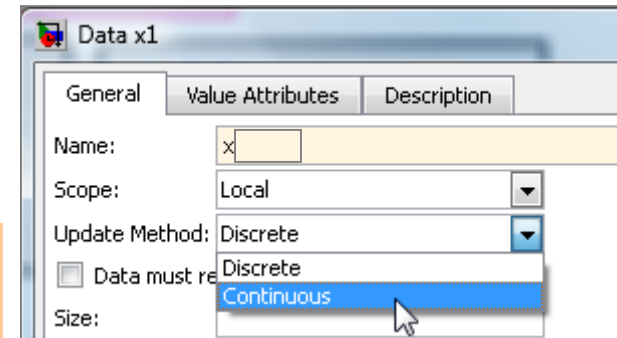
Hauptursache für Stateflow-Fehlermeldungen ist eine unpräzise Festlegung der Datenart für Konstanten, Parameter, diskrete und kontinuierliche Variablen, usw..



**zu a) kontinuierliche Zustandsgrößen = Integrierergrößen.****Beispiel:** Ort und Geschwindigkeit einer Masse

Kontinuierliche Zustandsgrößen werden durch numerische Integration berechnet (*during-Aktion*).
Sie ersetzen die Simulink-Integrierer.

Das Anlegen einer kontinuierlichen Zustandsgröße (z.B. x) **legt implizit und unsichtbar auch deren Ableitung** mit an (z.B. \dot{x}).



Sie dürfen nur in der Bedingungsaktion einer Transition verändert werden.

Sie dürfen nicht verändert werden :
- in der **during-Aktion**
- in der **Transitionsbedingung**

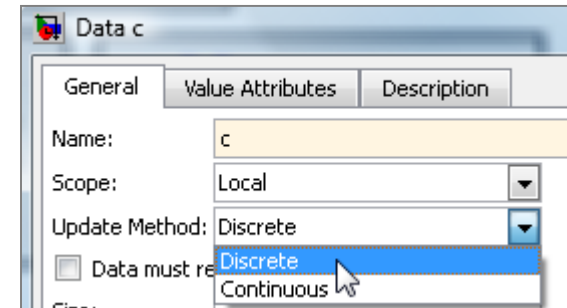
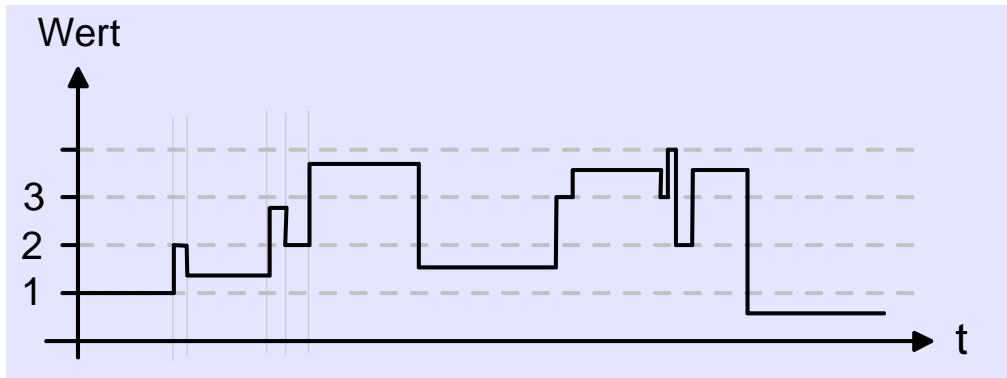
Hintergrund: Eine schlagartige Änderung einer Zustandsgröße x würde bedeuten, dass die Ableitung unendlich sein müsste.

Beispiel: Eine bewegte Masse kann nicht schlagartig die Geschwindigkeit ändern oder die Richtung umkehren (z.B. springender Ball).
Hierfür wäre eine unendliche Beschleunigung notwendig.



zu a) diskrete Zustandsgrößen = Umschaltgrößen, also Größen, die abschnittsweise konstant sind.

Beispiel: Fadenpendel mit Anschlag → die Fadenlänge



Sie dürfen nur in der Bedingungsaktion einer Transition verändert werden.

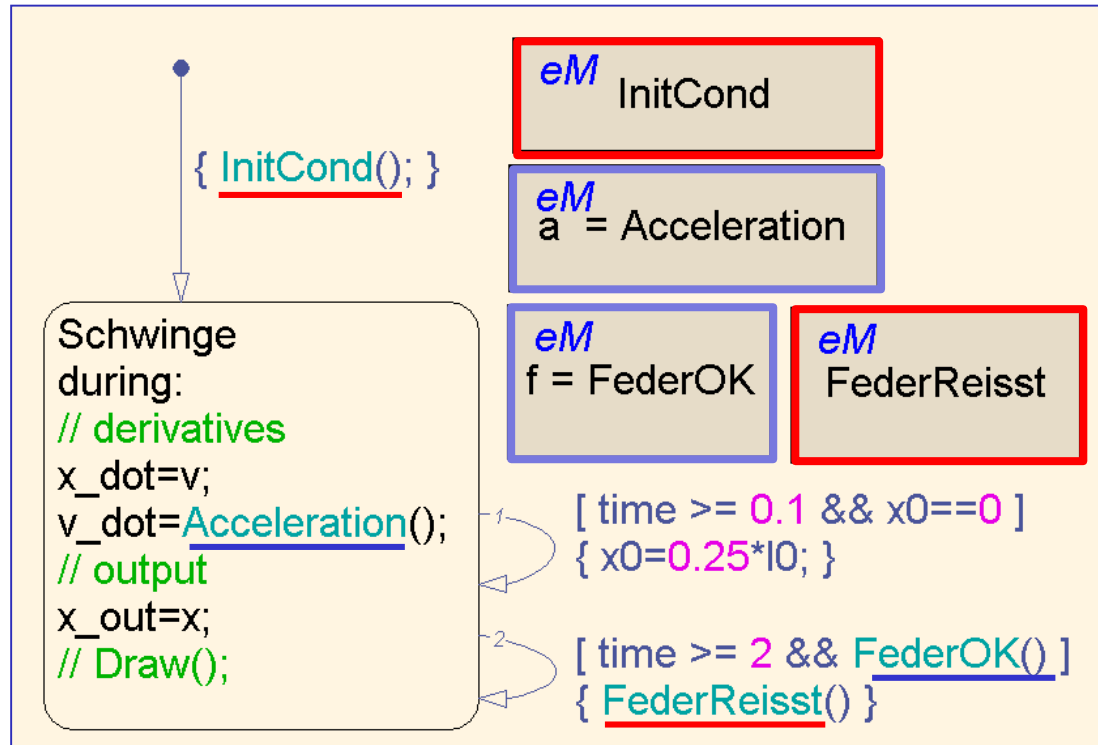
Sie dürfen nicht verändert werden :

- in der **during-Aktion**
- in der **Transitionsbedingung**

Hintergrund: Die Umschaltzeitpunkte lassen sich nur in Transitionsbedingungen exakt bestimmen.



Zusammengefasst: Wo gelten welche Schreibrechte auf Zustandsgrößen?



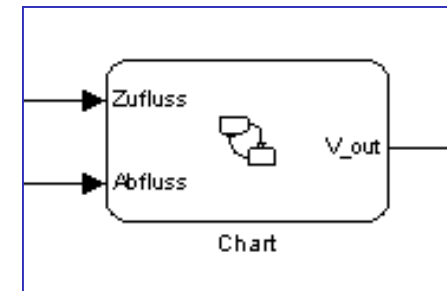
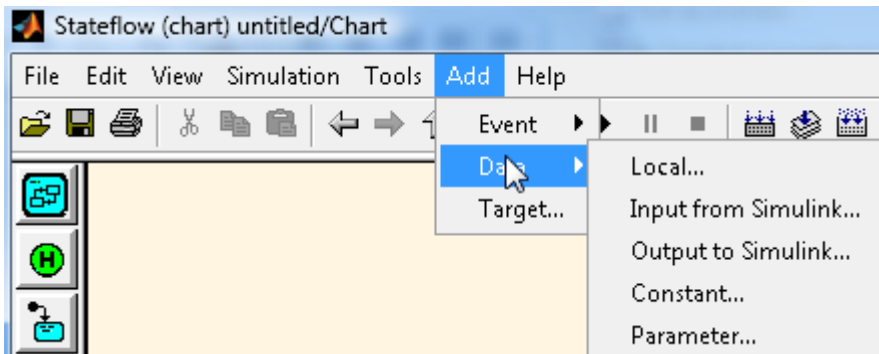
Schreibzugriff auf Zustandsgrößen ist nur hier erlaubt.

Hier dürfen nur die Funktionsrückgabewerte berechnet werden.
Kein Schreibzugriff auf Zustandsgrößen!

In allen Funktionen kann lesend auf Zustandsgrößen, Parameter, Eingangsgrößen und Konstanten zugegriffen werden.



zu b) und c) Ein- und Ausgangsgrößen von/nach Simulink



Input- und Output-Größen erzeugen entsprechende Konnektoren am Chartblock (hier z.B. Zufluss, Abfluss und V_{out}).

**zu d) Konstanten**

Größen, die während der Simulation konstant bleiben.

Nur Konstanten dürfen die Vektordimension anderer Größen festlegen.

Beispiel: Die Anzahl der Federn/Massen in einem Feder-Masse-System.

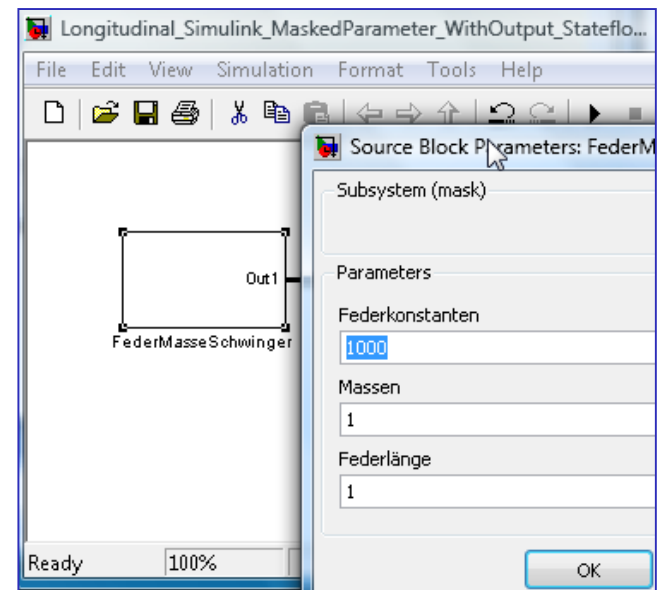
zu e) Parameter

Größen deren Einfluss auf die Simulation man untersuchen möchte.

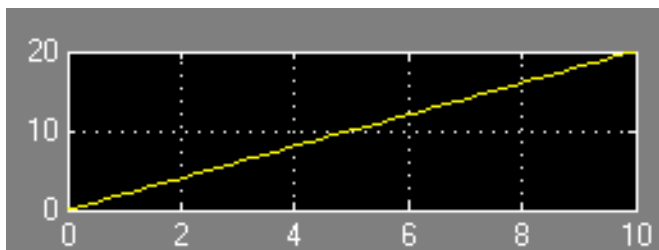
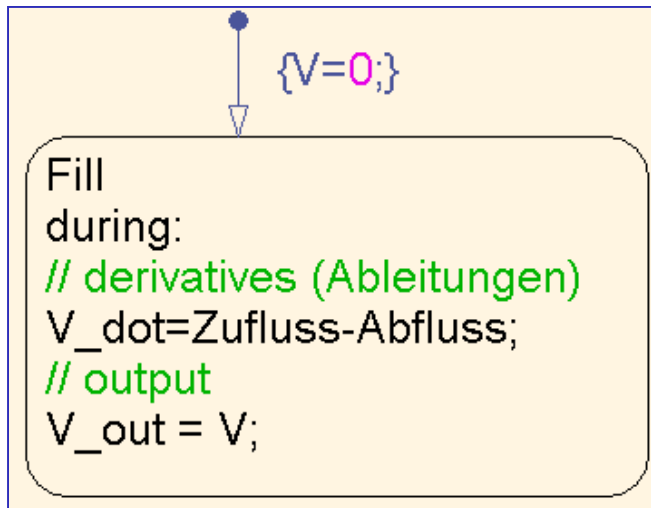
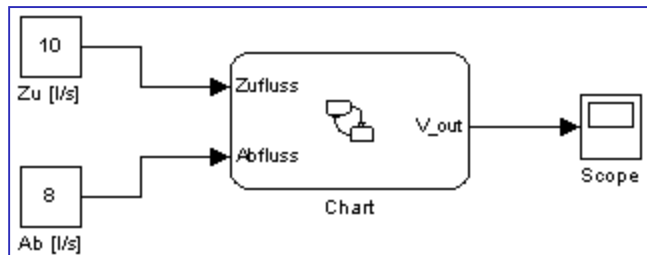
Beispiel: Einfluß von Länge und Masse eines Pendels

Parameter können später komfortabel geändert werden.

Ein Doppelklick auf das Teilsystem (welches den Zustandsautomat enthält) öffnet den Parameterdialog.

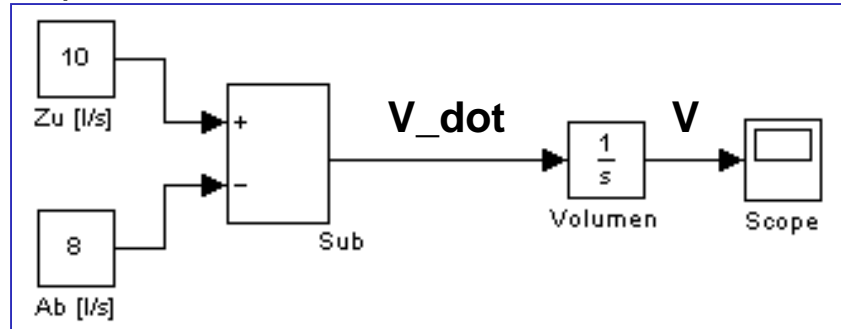


6.4.2.6 Modellierung dyn. Systeme mit Stateflow



Beispiel: Wasserbehälter mit Zu- und Abfluss

äquivalent zu



Die Zustandsgröße V wird mit „Add Data \rightarrow Local \rightarrow Continuous“ angelegt.

Implizit (und unsichtbar) wird dann auch die Ableitung der Zustandsgröße (hier V_dot) angelegt.

Zufluss und *Abfluss* sind Input-Größen.

V_out ist eine Output-Größe.

6.4.3 Modellierung hybrider Systeme mit Stateflow am Beispiel

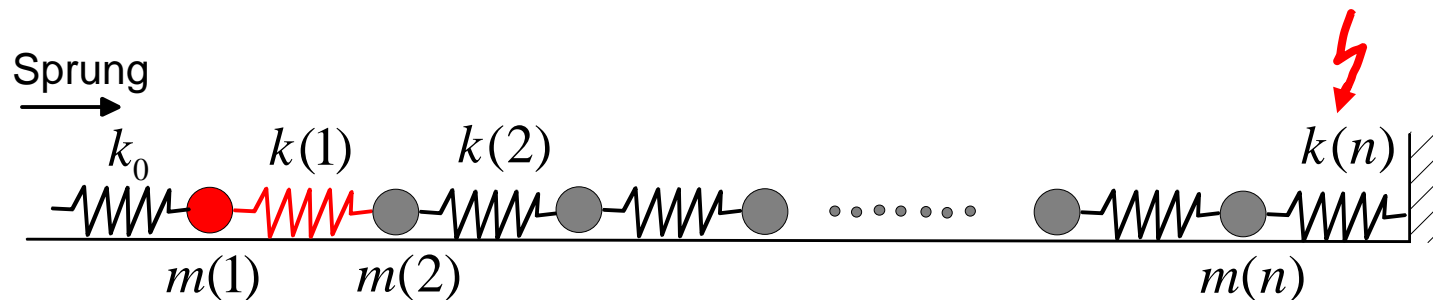
6.4.3.1 Beispiel : „Feder-Masse-Schwinger mit Federbruch“

n_Federn_Massen_Longitude_Stateflow_FastPlot_3

Modelliert werden soll ein dyn. System aus n Massen und n+1 Federn.

Zum Zeitpunkt $t=0.1s$ wird die Feder k_0 um $0.25 \cdot l_0$ (Federlänge) zusammengedrückt.

Zum Zeitpunkt $t=2s$ reißt die Feder $k(n)$, d.h. $k(n)=0$.



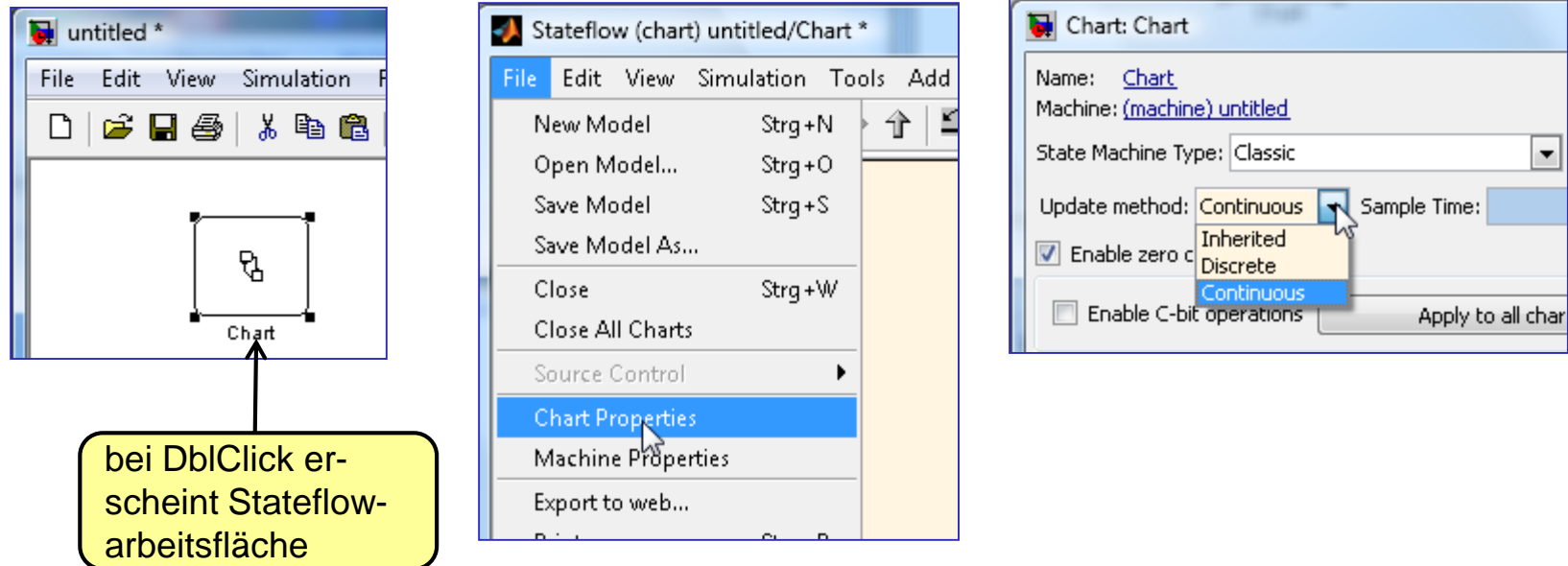
Die Beschleunigung der Masse m_i wird beschrieben durch folgende Beziehungen:

$$F_i = -[x_i - x_{i-1} - l_0] \cdot k_{i-1} + [x_{i+1} - x_i - l_0] \cdot k_i \quad a_i = \frac{F_i}{m}$$

Herleitung → Tafel

6.4.3.2 Modellierung der Simulation

a) Update-Method → *Continuous*

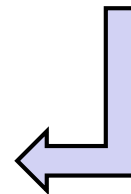
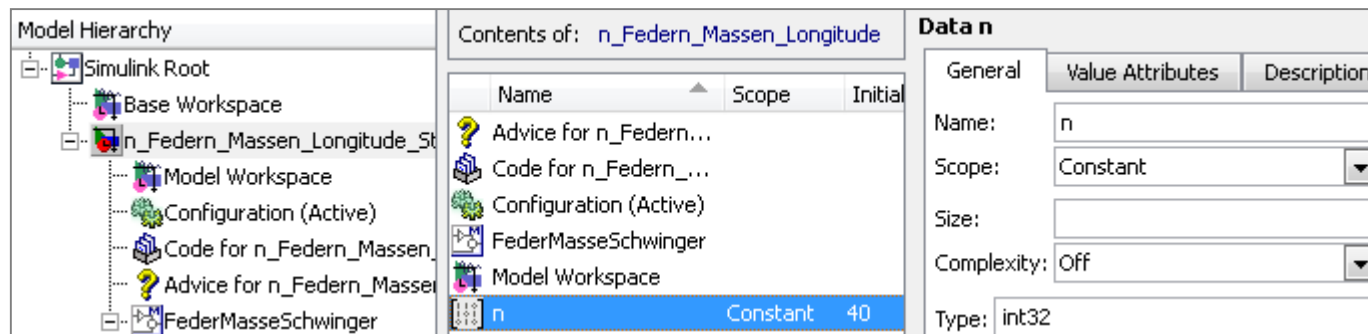
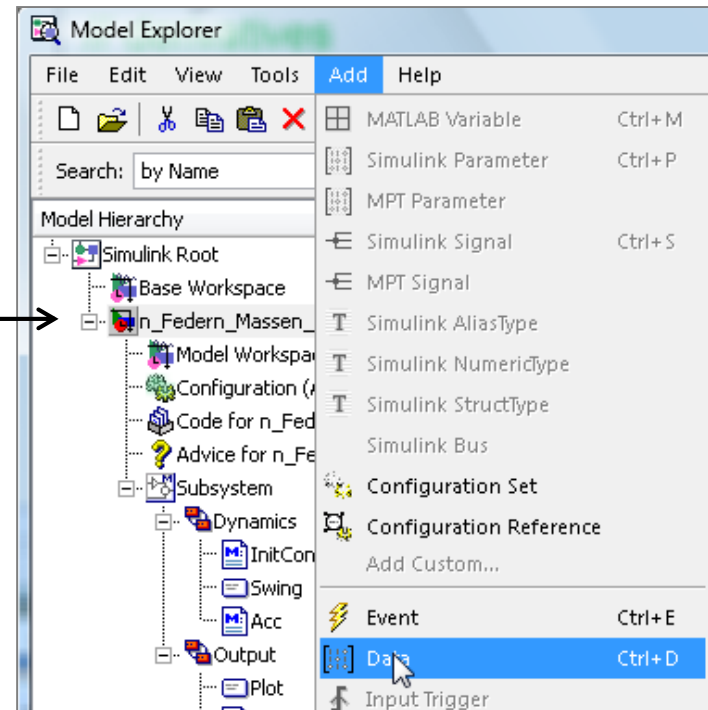
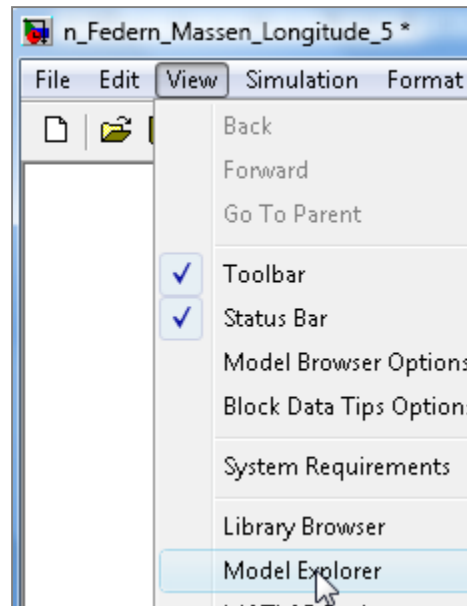


Damit die during-action des Zustandsdiagramms bei jedem Integrationsschritt durchlaufen wird und jederzeit Ereignisse ausgelöst werden können, muss die Update-Methode des Zustandsdiagramms auf *Continuous* gesetzt werden.



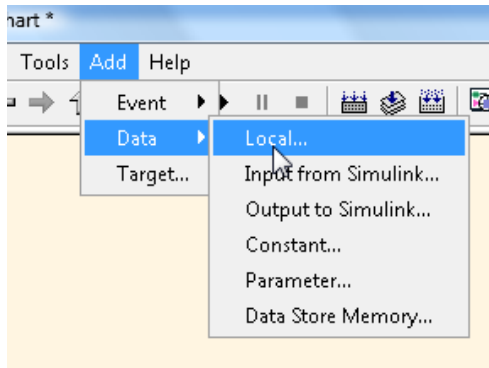
b) Konstante n „Anzahl der Massen“ anlegen

In diesem Fall soll die Konstante nicht nur dem Zustandsautomat bekannt sein, sondern modellweit zugreifbar sein.

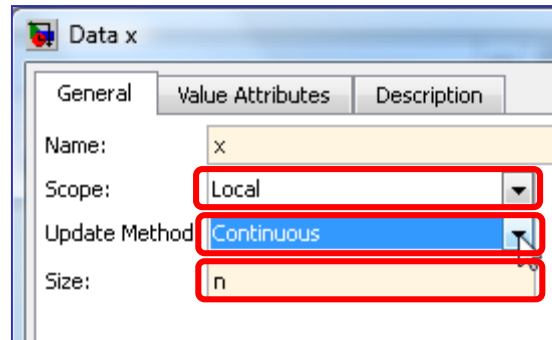


c) Zustandsgrößen (Integrierergrößen) anlegen

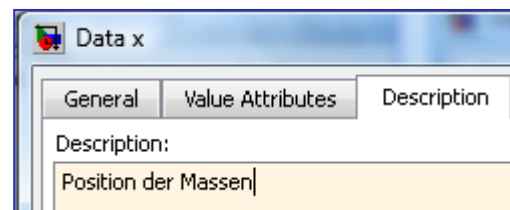
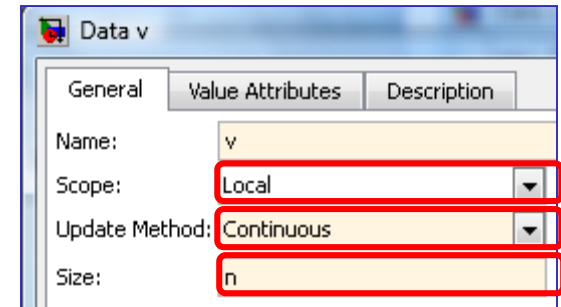
Zustandsgrößenvektor (Ort x und Geschwindigkeit v der Massen) anlegen.



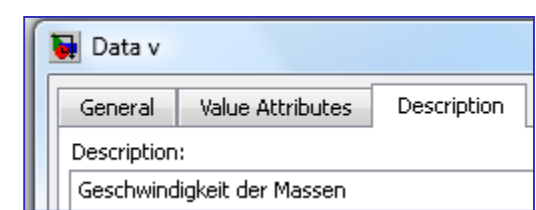
Ort der Massen



Geschwind. der Massen



Hier wird implizit die Ableitung **x_dot** mit angelegt !



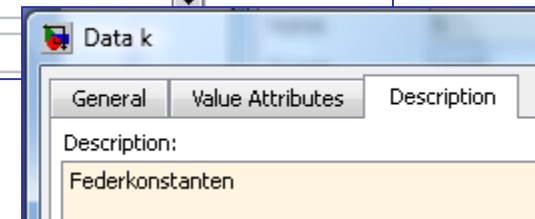
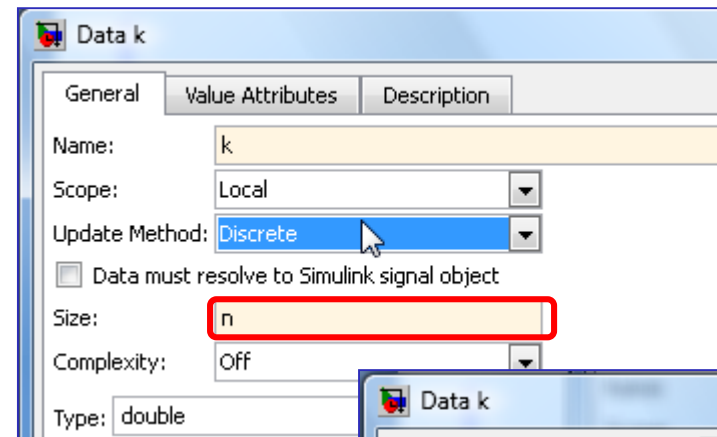
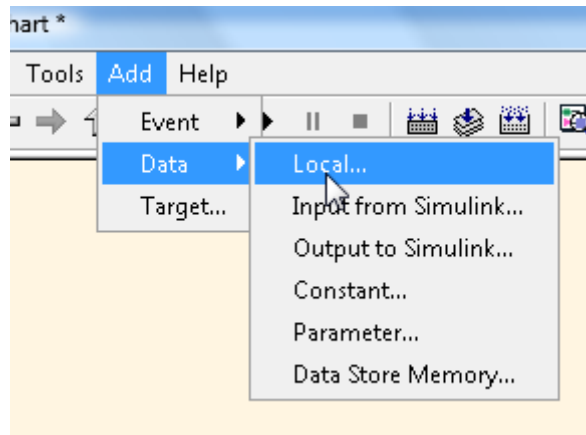
Hier wird implizit die Ableitung **v_dot** mit angelegt !

Hier werden n Integrierer $x(1) \dots x(n)$ und n Integrierer $v(1) \dots v(n)$ angelegt !



d) Zeitdiskrete Größen anlegen

Abschnittsweise konstante Größen (zeitdiskrete Größen / Umschaltgrößen) werden als „Add Data → Local → Discrete“ angelegt.



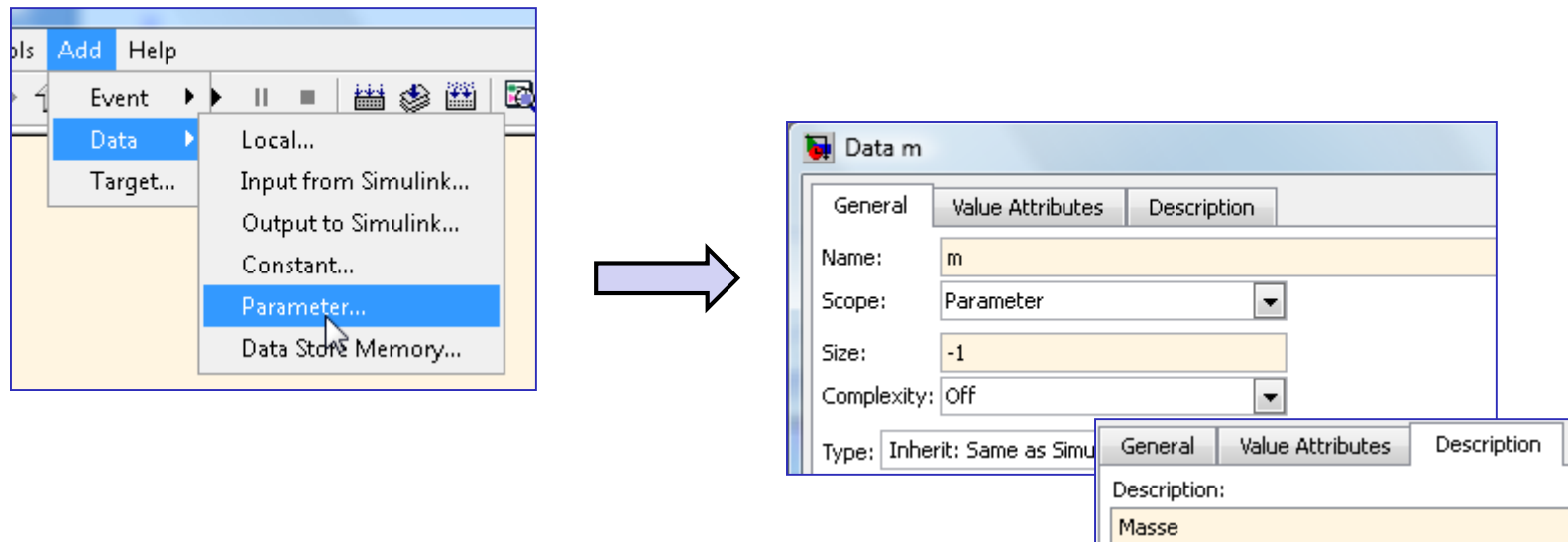
Desweiteren werden so angelegt: (s. S.77)

- k : Vektor von Federkonstanten
- x0 : Position des Eingangsadapters
- xend : Endposition der letzten Feder
- k0 : Federkonstante der ersten Feder



e) Parameter anlegen

Größen, die als Maskenparameter vorgebar sein sollen, werden mit „Add Data → Parameter“ angelegt.



So angelegt werden:

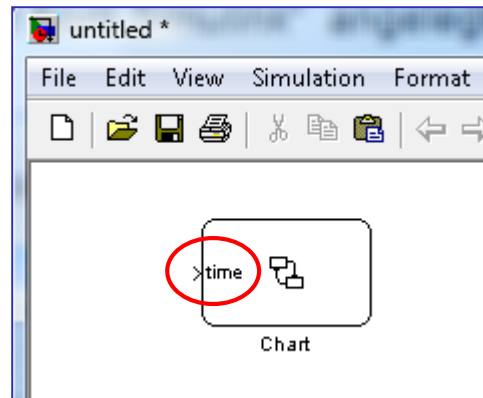
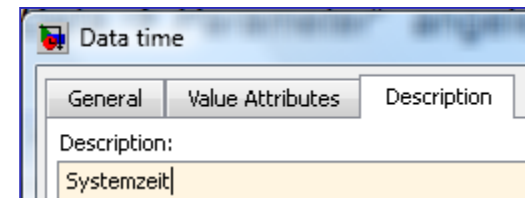
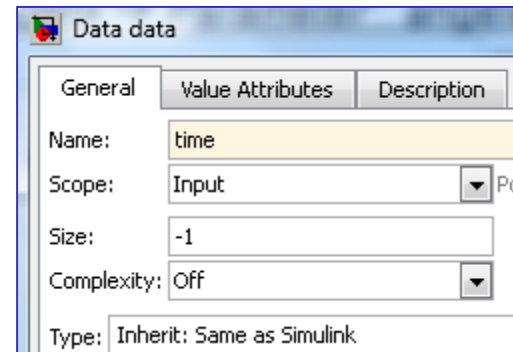
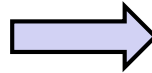
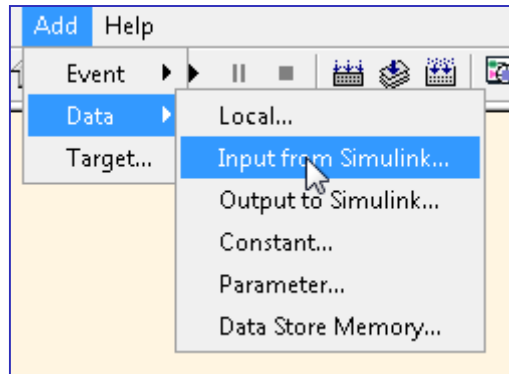
m : Einzelmasse (kg)

l0 : Federlänge der entspannten Feder

k_init : Initialwert der Federkonstanten

f) Input anlegen

Größen, die von Simulink über einen Signalpfad zugeführt werden sollen, werden mit „Add Data → Input from Simulink“ angelegt.

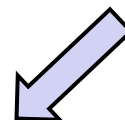
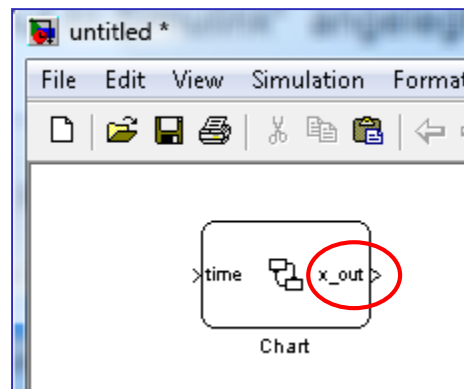
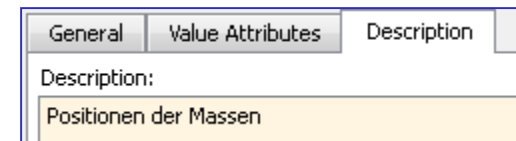
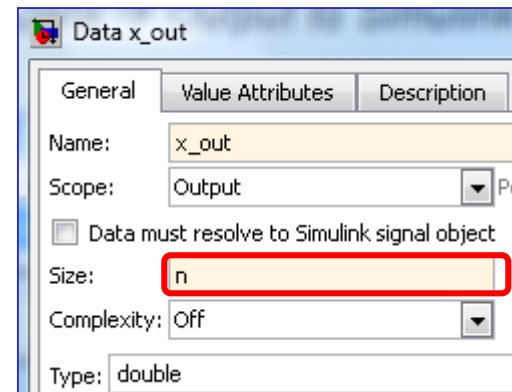
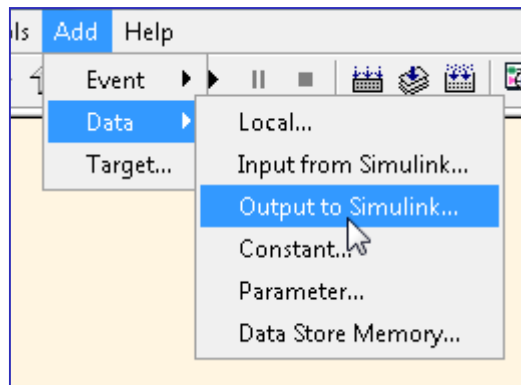




g) Output anlegen

Größen, deren Zeitverlauf untersucht und dargestellt werden soll, werden mit „Add Data → Output to Simulink“ angelegt.

Die Zustandsgrößen (hier $x(n)$ und $v(n)$) können nicht direkt ausgegeben werden!





Übersichtliche Darstellung aller Größen im Model-Explorer

Model Explorer

File Edit View Tools Add Help

Search: by Name Name: Search

Model Hierarchy

- Simulink Root
 - Base Workspace
 - n_Federn_Massen_Longitude_Stateflow_FastPlot_3
 - Model Workspace
 - Configuration (Active)
 - Code for n_Federn_Massen_Longitude_Stateflow_FastPlot_3
 - Advice for n_Federn_Massen_Longitude_Stateflow_FastPlot_3
 - FederMasseSchwinger
 - Chart
 - Schwinge
 - InitCond
 - Acceleration
 - FederOK
 - FederReisst

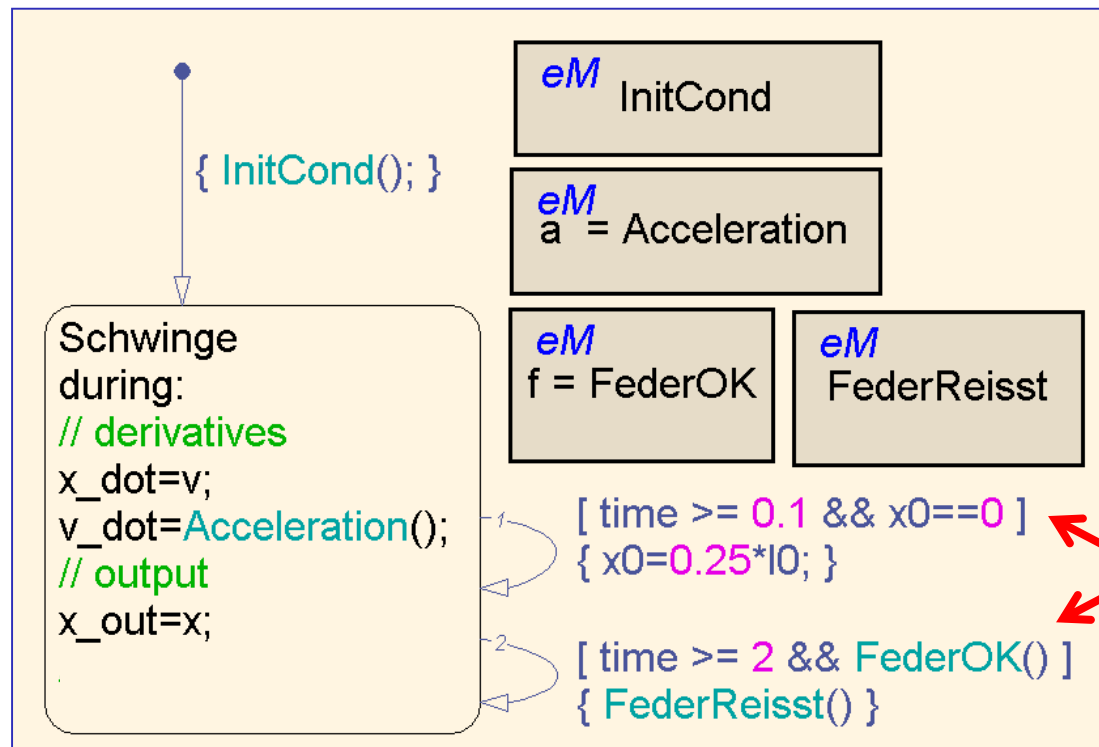
Contents of: n_Federn_Massen_Longitude_Stateflow_FastPlot_3/FederMasseSchwinger

Name	Scope	UpdateMethod	DataType	Size
k	Local	Discrete	double	n
k_init	Parameter	Discrete	Inherit: Same as Si...	-1
k0	Local	Discrete	double	1
l0	Parameter	Discrete	Inherit: Same as Si...	-1
m	Parameter	Discrete	Inherit: Same as Si...	-1
time	Input	Discrete	Inherit: Same as Si...	-1
v	Local	Continuous	double	n
x	Local	Continuous	double	n
x_out	Output	Discrete	double	n
x0	Local	Discrete	double	
xend	Local	Discrete	double	



h) Automat erstellen

Aktionen (und evtl. auch Bedingungen) in den Zustandsübergängen und in den Zuständen fasst man am sinnvollsten in Embedded-Matlab-Funktionen zusammen. Die Differentialgleichungen sowie die Übergabe der Ausgabewerte werden in die *during*-Section eingetragen.



Zeitveränderliche Größen nicht auf Gleichheit abgefragt (z.B. `time == 0.1`), da man nicht sicherstellen kann, dass die Größe exakt diesen Wert einnimmt.

Die Übergangsbedingungen müssen so formuliert werden, dass der Übergang nur einmal erfolgt.

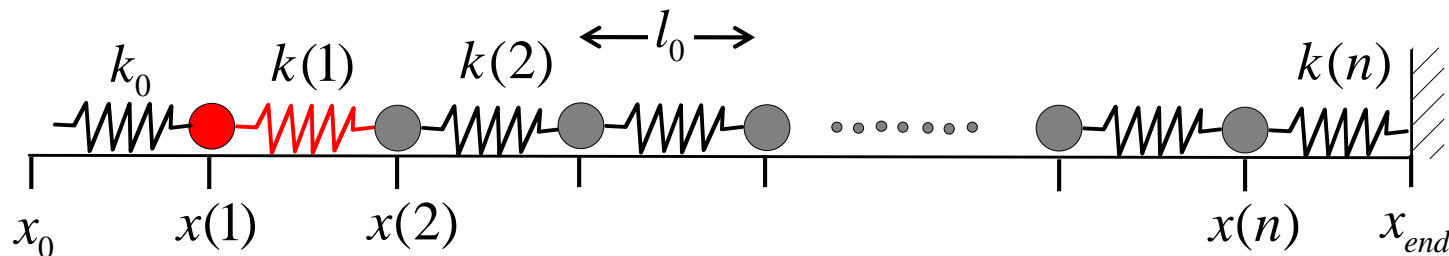


```
function InitCond
    % a) Die k Massen auf die Startpositionen setzen.
    %   Abstand zweier Massen: 10
    % b) Federkonstanten initialisieren
    for i = 1:n
        x(i) = 10*i;
        k(i) = k_init;
    end

    x0 = 0;           % Ort des Eingangspunktes
    xend = double(10*(n+1)); % Ort des Endpunktes
    k0 = k_init;      % Federkonstante der ersten Feder
end
```

Lesezugriff auf : Zustandsgrößen, Umschaltgrößen, Parameter, Konstanten

Schreibzugriff auf : Zustandsgrößen, Umschaltgrößen



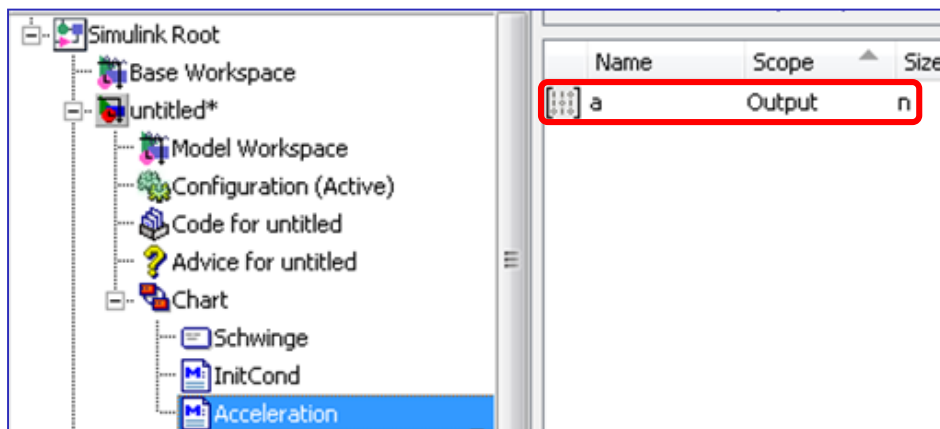


```
function a = Acceleration

% Achtung: Size des Rückgabeparameters a muss im
% ModelExplorer explizit auf n gesetzt werden !!!
a=zeros(n,1);

% Die erste und die letzte Masse müssen gesondert
% behandelt werden
a(1)=( -(x(1)-x0) -10) *k0      + (x(2)-x(1)-10) *k(1) ) / m;
a(n)=( -(x(n)-x(n-1)-10) *k(n-1) + (xend-x(n)-10) *k(n) ) / m;

% Kraftwirkung zwischen den Massen 2 ... n-1
for i=2:n-1
    a(i) = ( -(x(i)-x(i-1)-10) *k(i-1) + (x(i+1)-x(i)-10) *k(i) ) / m;
end
```



Lesezugriff auf :

- Zustandsgrößen,
- Umschaltgrößen,
- Parameter und Konst.

Schreibzugriff nur auf :

- Rückgabeparameter



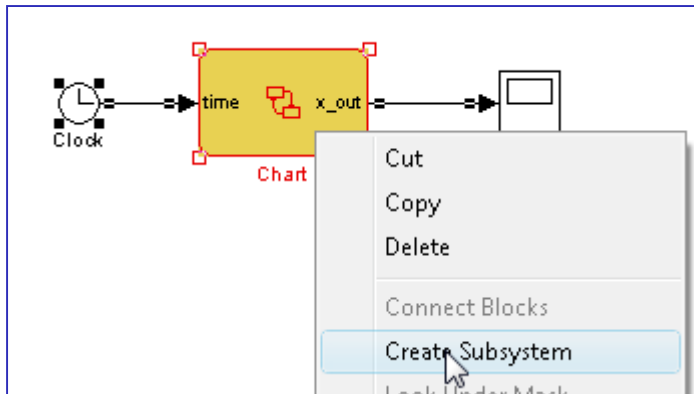
```
function f=FederOK()  
    f=true;           % Muss im ModelExplorer explizit  
                     % auf Boolean gesetzt werden !  
    if k(n)== 0      % Wenn k(n)==0, dann ist Feder defekt  
        f=false;  
    end  
end
```

Lesezugriff auf : Zustandsgrößen, Umschaltgrößen, Parameter, Konstanten
Schreibzugriff **nur auf** : Rückgabeparameter

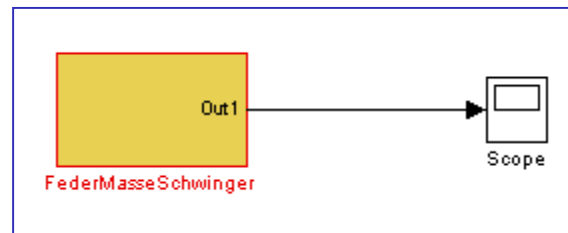
```
function FederReisst  
% Feder defekt -> Federkonstante = 0  
    k(n)=0;  
end
```

Lesezugriff auf : Zustandsgrößen, Umschaltgrößen, Parameter, Konstanten
Schreibzugriff auf : Zustandsgrößen, Umschaltgrößen

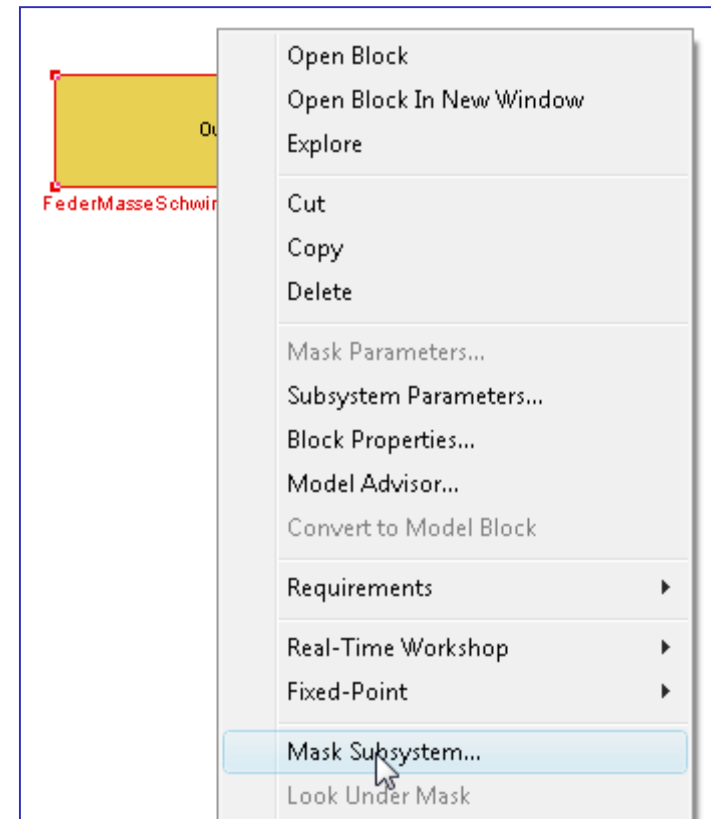
i) Eingabemaske für Simulationsparameter erstellen

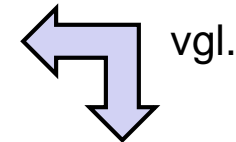
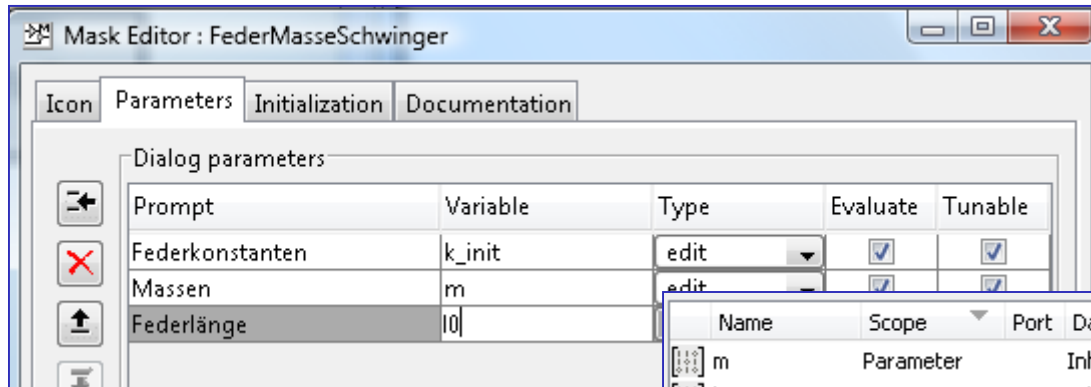





erzeugt aus den markierten Elementen ein Subsystem



Mit „Mask Subsystem“ wird eine Parametereingabemaske erstellt.

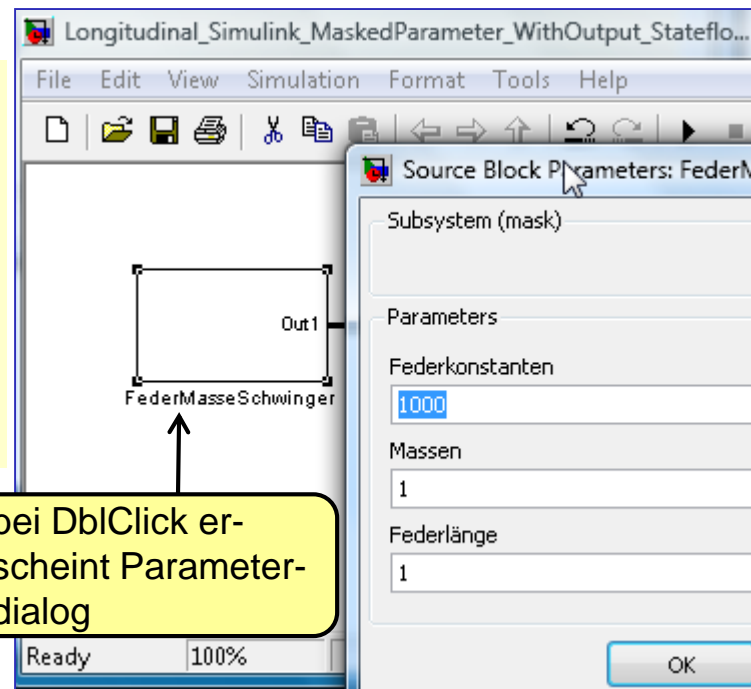




	Name	Scope	Port	DataType	Size	UpdateMethod
	m	Parameter		Inherit: Same as Simulink	-1	Discrete
	l0	Parameter		Inherit: Same as Simulink	-1	Discrete
	k_init	Parameter		Inherit: Same as Simulink	-1	Discrete

Wird jetzt auf das Subsystem geklickt, so erscheint der Parameterdialog.

Das Subsystem lässt sich öffnen mit dem Kontextmenü „Look under Mask“.

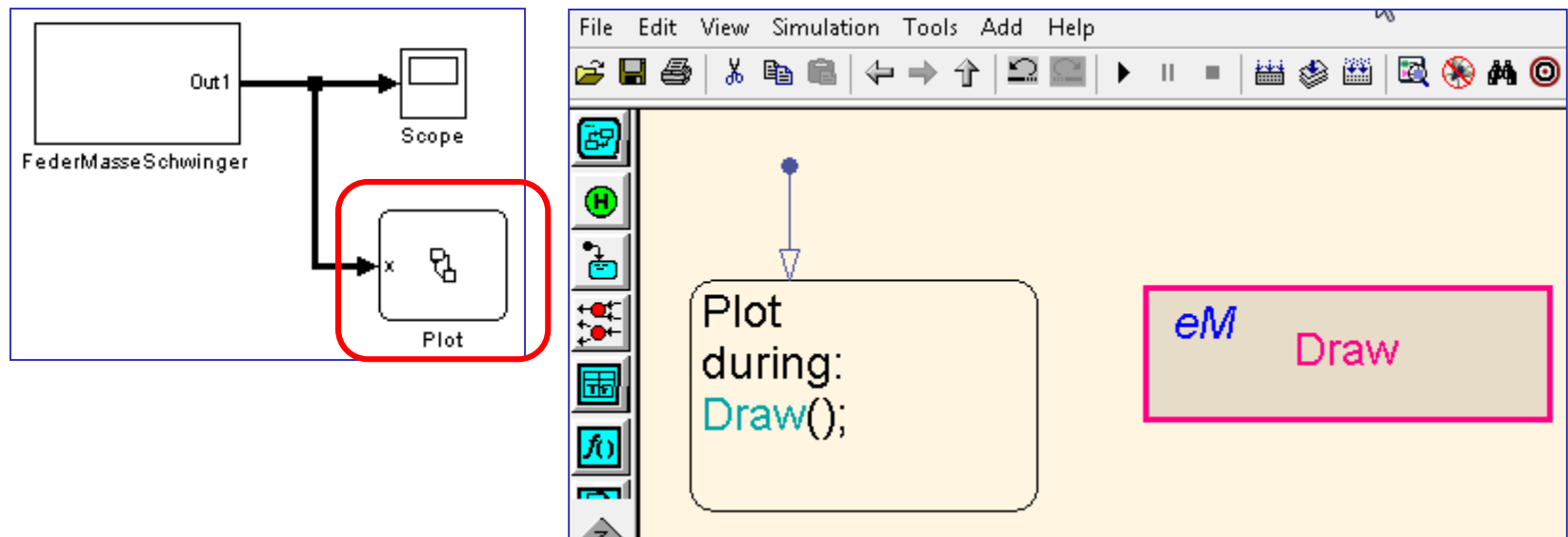


bei DbIClick erscheint Parameterdialog

j) Grafische Ausgabe realisieren

Die grafische Ausgabe wird in einem zweiten Automaten (**update**: inherited) realisiert.
Grund: Die *during*-Actions des kontinuierlichen Automaten werden bei jedem Integrationsschritt durchlaufen. Das passiert weit häufiger als für die Ergebnisausgabe nötig ist.

Eine Zeichenfunktion im *during*-Bereich des kontinuierlichen Automaten würde daher die Ausgabe erheblich und unnötig verlangsamen.



x als *Input* hinzufügen (discrete, firstIndex=1).

Die Vektordimension von x ergibt sich aus der Verbindung (size=-1)



```
function Draw

    eml.extrinsic('set', 'drawnow', 'scatter', 'gca');

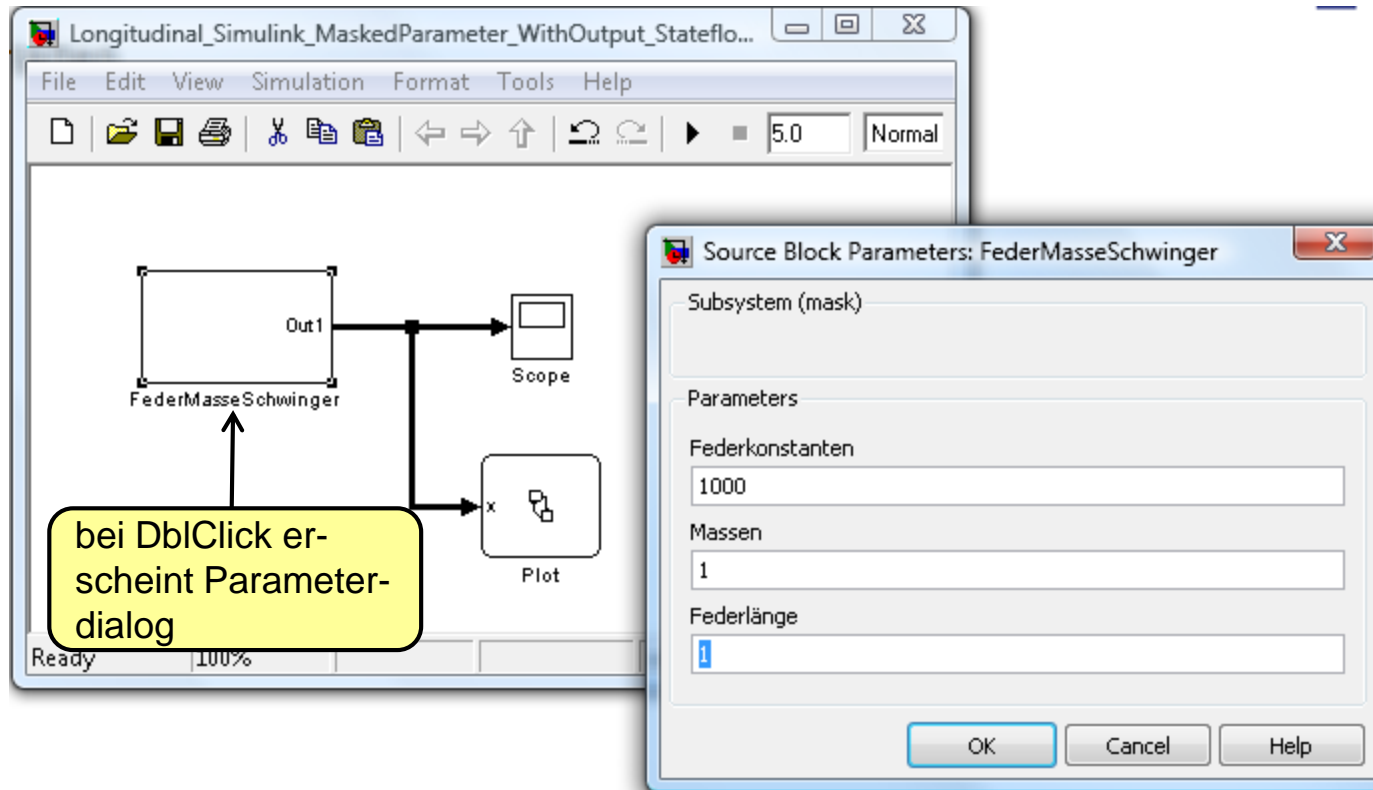
    y=zeros(1,n);

    % Scatterplot zeichnen
    scatter(x', y, 'o', 'MarkerFaceColor', [1 0 0]);
    set (gca,'XLim', [0 (n+1)*10] ); % x-Bereich festhalten
    drawnow;                        % jetzt zeichnen
end
```

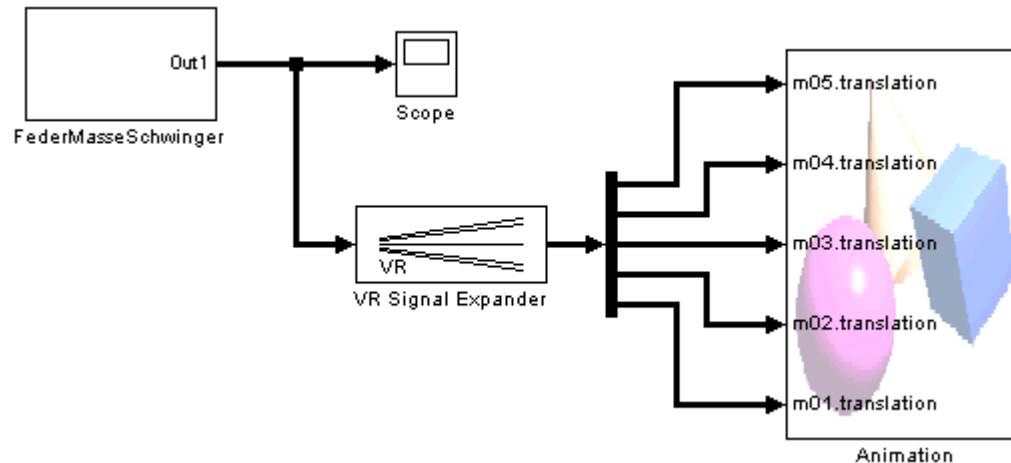


k) Simulation starten `n_Federn_Massen_Longitude_Stateflow_FastPlot_3`

Jetzt können die Simulationsparameter angegeben und die Simulation gestartet werden.



I) Alternative Visualisierung mit Virtual-Reality-Block



Nachteil der Lösung:

Die Anzahl der Massen ist nicht so leicht änderbar. Für eine andere Masseanzahl muss eine andere virt. Welt erzeugt werden.

