Commando Window Befehle

`clear`	Workspace/Variabeln wird gelöst
`close`	Figures werden geschlossen
`clc`	Command Window wird geleert
`% …`	Kommentar
`vpa(x,n)`	x auf n Stellen runden

Matrizen und Vektoren

`M = [1,2,3;4,5,6]`	2 Zeilen, 3 Spalten
`M = [1 2 3; 4 5 6]`	2 Zeilen, 3 Spalten
`a = M(2,1)`	Element 2. Zeile, 1. Spalte
`a = M(1, :)`	alle Elemente der 1. Zeile
`[a, b] = M(3, 1:2)`	Element 1 bis 2 der 3. Zeile in a, b Speichern
`M(1, 1) = []`	Element 1. Zeile, 1. Spalte löschen
`M(:, 2) = []`	alle Elemente der 2 Spalte löschen

`t=[start:step:end]`

`eye(n)`	`eye(n,m)`	Einheitsmatrix mit dim n
`ones(n)`	`ones(n,m)`	Matrix mit Einträgen 1
`zeros(n)`	`zeros(n,m)`	Matrix mit Einträgen 0
`rand(n)`	`rand(n,m)`	Matrix mit Zufallswerten zwischen 0 und 1

n Zeilen m Spalten

Operatoren

`* ^ \`	Vektor-, Matrixoperatoren
`.* .^ \`	Elementweise Operatoren
`Matrix.'`	Transponierte
`transpose(M)`	Transponierte

`min(v), max(v)`	Minimum, Maximum Wert
`mean(v)`	Mittelwert
`sum(v)`	Summe der Elemente
`inv(m)`	Inverse
`abs(n)`	Betrag von n
`numel(A)`	#Elemente in Matrix A
`exp(x)`	e^x
`AB = C`	
• `A=C/B`	$A=C*B^{-1}$
• `B=A\C`	$B=A^{-1}*C$
`length(v)`	Anzahl Elemente in v
`size(v)`	Zeilen Spalten

Logische Operatoren

`A < B, A <= B`	A kleiner (gleich) B
`A == B, A ~= B`	A (nicht) gleich B
`A & B, A | B`	UND, ODER
`~A`	NICHT A
`xor(A, B)`	Exklusiv ODER
`logical([1 0 1])`	Logical Array

Logischer Output {0, 1}

Char-Arrays und Strings

`t = 'Hello, world'`	Char-Array, kann indexiert werden
`[t,t]`	Char-Array zusammenhängen
`t = „Hallo World "`	String, gilt als einzelnes Element
`disp('bla bla')`	Anzeigen von Text in CW

Graphische Funktionen

`figure(num)`	Erzeugt ein Fenster, spricht ein Fenster an, wenn es schon existiert
`close num`	Schliesst die jeweilige Figure
`close all`	Schliesst alle Figuren
`set(gcf, 'Color', [1 1 1])`	Hintergrundfarbe in RGB

get current figure

`plot(x, y)`	
`plot(t, [y1;y2])`	mehrere Plots zu t
`plot(t1, y1, t2, y2)`	mehrere Plots mit unterschiedlichen t

Plot Manipulationen

`plot(t, y, 'bo--')`	Punkte o, Linie gestrichelt, Farbe blau

`plot(t1, y1, 'ro--', t2, y2, 'm+-.')`

b	blue	.	point	-	solid
g	green	o	circle	:	dotted
r	red	x	x-mark	-.	dashdot
c	cyan	+	plus	--	dashed
m	magenta	*	star	(none)	no line
y	yellow	s	square		
k	black	d	diamond		
w	with	v	triangle (down)		
		^	triangle (up)		
		<	triangle (left)		
		>	triangle (right)		
		p	pentagram		
		h	hexagram		

`plot(x,y,'ParameterName1',ParameterWert1 ,'ParameterName2',ParameterWert)`

- `'LineWidth'` 0.5(Default), 1, 1.5
- `'Color'` [0, 0.5, 0.5] (RGB Triplet)
- `'LineStyle'` '-','-.',':','--'

Beschriftungen

`xlabel('Text')`	x-Achsen Beschriftung
`ylabel('Text')`	y-Achsen Beschriftung
`title('Text', 'FontSize', 14)`	Plot Titel mit Schriftgrösse
`axis([0 15 -1.5 1])`	Achsen von [x_{min}, x_{max}, y_{min}, y_{max}]
`grid`	Raster EIN
`hold on, hold off`	Mehrere Plots

```
t = [0:0.5:10];
plot(t, sin(t));
hold on
plot(t, 0.5*sin(t));
```

`subplot()`

```
Subplot(3, 2, 1);      3 Zeilen, 2 Spalten, Plot 1
Plot(t, sin(t));
Subplot(3, 2, 2);      3 Zeilen, 2 Spalten, Plot 2
plot(t, 0.sin(2*t));
```

I/O Operationen

`save meineDaten`	Gesamter Workspace im current Folder speichern
`save meineDaten Var1 Var2`	Bestimmte Variabeln im current Folder speichern
`load meineDaten`	Daten im current Folder in den Workspace laden
`xlswrite('ExelBsp.xls', A)`	Schreibt A in ein Exel File

Laden von Exeldaten

```
[num,text,rohdaten] = xlsread('ExelBsp.xls')
```

Funktionen

```
function rueckgabewert = funcname(v1,n1)
        …
        rueckgabewert = sum(v1)/n1;
end
```

if … elseif … else

```
a = 33;
b = 200;
if b > a
    disp('a < b')
elseif b == a
    disp('a == b')
else
    disp('a > b)
end
```

while - for

```
fruits = ["apple","banana","cherry"]]
for x = 1 :length(fruits)
    if fruits(x) == "banana"
            continue
    elseif fruits(x) == "cherry"
            break
    end
    disp(fruits(x))
end
```

```
i = 1
while i < 6
    disp(i)
    i = i+1 ;
end
```

Switch … Case … Otherwise

```
x = input('Give name : ', 's')
switch x
    case 'John'
            disp('Hi John')
    case 'Joe'
            disp('Bye Joe')
    otherwise
            disp('☹')
end
```

Symbolic Math Toolbox

`v = sym('v');`	Erstellt eine symbolic Variable
`syms x y v`	Erstellt symbolic Variabeln

Plot

`fplot(F1,[0 10],'b')`	symbolic plot, alle Befehle von plot verfügbar.

```
t = [0:0.1:10]
F2 = double(subs(F1, x, t))
```

Ersetzt das symbolic x mit den Werten von t und konvertiert zu double (für plot(t, F2))

`subs(f, {a,b,c}, {1,2,[0:0.1:6]})`	Variabeln einsetzen
`expand(f)`	Polynom ausmultiplizieren
`simplify(f)`	vereinfachen

Differentation

```
dfy = diff(f, y, 1)
```
nach symbolic y ableiten

```
dfnum = diff(f)/dt
```
numerisch ableiten

```
      = diff(f,t)
```
Schrittgrösse dt

```
pretty(f)
```
Funktion übersichtlicher darstellen

```
int(f,x,[0 1])
```
bestimmtes Integral [0, 1]

Lösen von Gleichungssystemen

```
solve(6*x^2 - 2*x+14 == 0, x)
solve(x^2-x^2*y+2*x*y^2-6*y^2 == 0,y)
```

```
A=[1,4;4,2]
B=[4;2]
X=A\b
```

numerisch

$$\begin{cases} x + 4y = 4 \\ 4x + 2y = 2 \end{cases}$$

Logik – Boolsche Algebra

∧ AND ∨ OR ¬ NOT ⊕ XOR

```
X= and(A, B)= A&B
```
AND

```
X= or(A, B)= A|B
```
OR

```
X= not(A)= ~A
```
NOT

```
X= not(and(A,B))= ~(A&B)
```
NAND, NOT AND

```
X= not(or(A,B))= ~(A|B)
```
NOR, NOT OR

```
X= xor(A,B)
```
XOR, exclusive OR ⊕

```
X= ~xor(A,B)
```
NXOR, NOT exclusive OR

```
wert_char = dec2bin(n)
```
n Dezimalzahl zu Binär

```
wert_log = logical(wert_char-'0')
```
Numerisch zu logischem Wert

```
T = truth_table(N)
```
N: Eingangsvariablen 2^N Zeilen

Erstellt Wahrheitstabelle

```
A=T(:,1) B=T(:,2)
```
Werte zuordnen

Nr	Gesetz	Rechenregel	
1	Identitätsgesetz	$A \cdot 0 = 0$	» A&0
2		$A \cdot 1 = A$	» A&1
3		$A \cdot A = A$	» A&A
4	Komplementgesetz	$A \cdot \bar{A} = 0$	» A&~A
5	Identitätsgesetz	$A + 0 = A$	» A|0
6		$A + 1 = 1$	» A|1
7		$A + A = A$	» A|A
8	Komplementgesetz	$A + \bar{A} = 1$	» A|~A
9		$\bar{\bar{A}} = A$	» ~(~A)
10	Kommutativgesetz	$A \cdot B = B \cdot A$	» A&B; B&A
11		$A + B = B + A$	» A|B; B|A

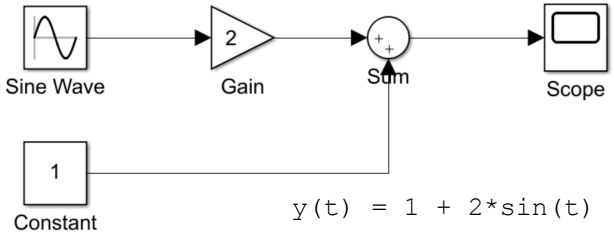
Nr	Gesetz	Rechenregel	
			» A=1; B=1; C=1; D=1;
12	Assoziativgesetz	$(A \cdot B) \cdot C = A \cdot (B \cdot C) = A \cdot B \cdot C$	» (A&B)&C; A&(B&C); A&B&C
13		$(A + B) + C = A + (B + C) = A + B + C$	» (A|B)|C; A|(B|C); A|B|C
14	Distributivgesetz	$(A \cdot B) + (A \cdot C) = A \cdot (B + C)$	» (A&B)|(A&C); A&(B|C)
15		$(A + B) \cdot (A + C) = A + (B \cdot C)$	» (A|B)&(A|C); A|(B&C)
16	Allg. Distributivgesetz	$(A + B)(C + D) = AC + AD + BC + BD$	» (A|B)&(C|D); A&C|A&D|B&C|B&D
17		$AB + CD = (A + C)(A + D)(B + C)(B + D)$	» A&C|C&D;(A|C)&(A|D)&(B|C)&(B|D)
18	Absorbtionsgesetz	$A \cdot (A + B + C) = A$	» A&(A|B|C)
19		$A + (A \cdot B \cdot C) = A$	» A|(A&B&C)
20	De Morgan	$\overline{A \cdot B} = \bar{A} + \bar{B}$	» ~(A&B); ~A|~B
21		$\overline{A + B} = \bar{A} \cdot \bar{B}$	» ~(A|B); ~A&~B

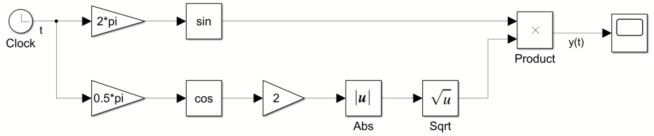
Logische Ausdrücke vereinfachen

```
Syms S1 S2 S3
X=simplify((~S1&S2&~S3)|(S1&~S2&~S3)|(S1&S2&~S3))
```

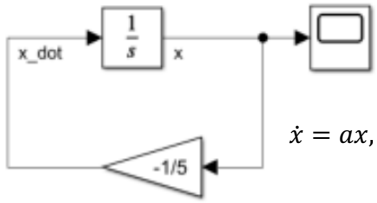
Simulink

```
simulink
```
Simulink starten



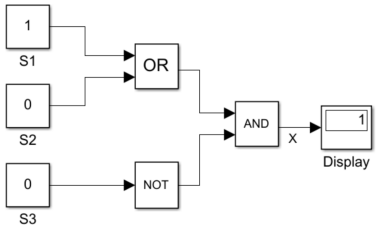


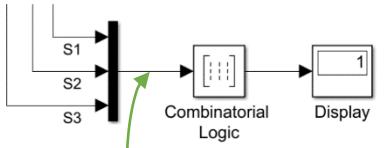
$$y(t) = \sin(2\pi t) \cdot \sqrt{\left|2 \cdot \cos\left(\frac{1}{2}\pi t\right)\right|}$$



$$\dot{x} = ax, \qquad a = -\frac{1}{5}$$

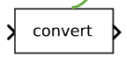
Logik im Simulink





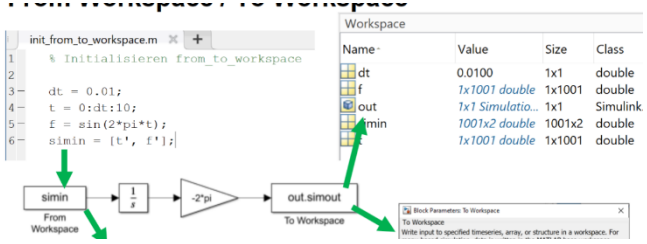
letzte Spalte Ausgabewerte

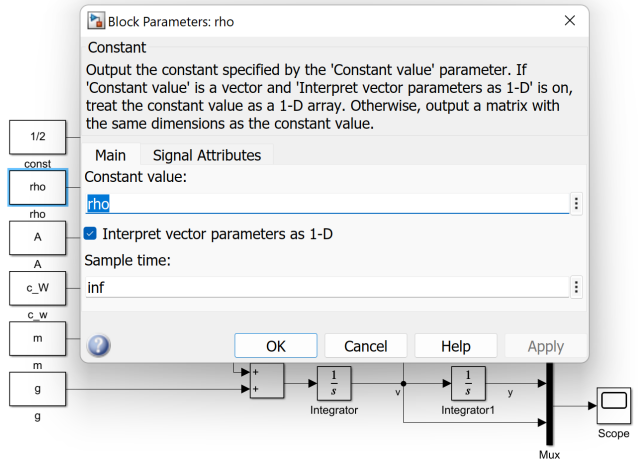
S1 Block Parameters → Output data type → **boolean**



Signale automatisch umwandeln

From Workspace/To Workspace – Übung 10

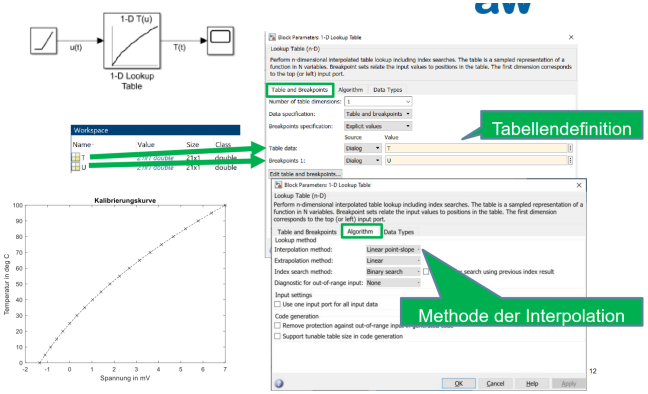




Constant Block.

Lookup Tables

Woche 11



Extremwertproblem

Minimieren des Umfangs

```
syms b h          % Definieren der symbolischen Variablen
A1=36;            % Flächeninhalt definieren
h=A1/b;           % Verhältnis zwischen Höhe h und Breite b
                  mittels Fläche: h=36/b

U=2*b+2*h;        % Umfang eines Rechtecks symbolisch
Ud=diff(U,b);     % Erste Ableitung von U nach b  (Ud==0 für
                  Minimum und Maximum)

b12=double(solve(Ud==0,b));    % Löse Ud==0 nach b

idx=b12>0;        % Alle Indizes von Breite b>0
b1=b12(idx)       % Wert der Breite b>0 und im Command-
                  Window anzeigen
h1=double(subs(h,b1))  % Breite in h=A/b einsetzen und im
                       Command-Window anzeigen
```

Maximieren des Flächeninhalts

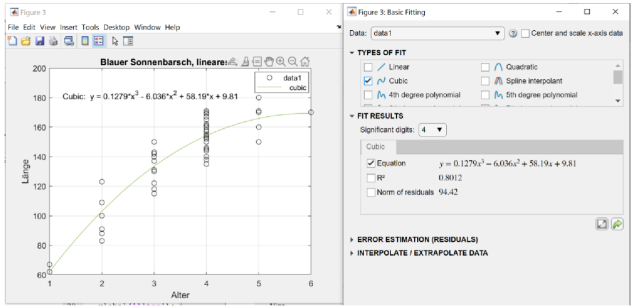
```
U2=64;  % Umfang definieren

h=(U2/2)-b;       % Verhältnis zwischen Höhe h und Breite b
                  mittels Umfang: h=(64/2)-b

A=b*h;  % Fläche eines Rechtecks symbolisch

Ad=diff(A,b);     % Erste Ableitung von A nach b  (Ad==0 für
                  Minimum und Maximum)

b2=double(solve(Ad,b))% Löse Ad==0 nach b und im
                       Command-Window anzeigen

h2=double(subs(h,b2))  % Breite in h=(64/2)-b einsetzen und
                       im Command-Window anzeigen
```

Lineare Regression

```
y=[1, 2, 1.3, 3.75, 2.25]';
x=[1, 2, 3, 4, 5]';

X=[ones(size(x)), x, x.^2]     Polynom Regression
b=X\y;                         für linear weglassen

x_new=6                        % Vorhersage an
X_pred=[1,x_new]               % Punkt x=6

plot(x, y, 'ko', x, X*b, 'r')
hold on
plot(x_new, X_pred*b, 'm*', 'MarkerSize', 12);
hold off
grid on
```

Interaktive Kurvenanpassung

In Figure → Tools → Basic Fitting



Vorsicht bei Extrapolation mit höherer Ordnung!!!

`mldivide`	«\» Operator
`polyfit`	Kurvenanpassung mit Polynomen
`cftool`	Interaktive Kurvenanpassung in Plots

Differenzialgleichungen

```
[t, y] = ode23(odefun, tspan, y0)
```

`odefun` «function handle» für die ODE y'=f(t,y)

`tspan` Zeitspanne [t_0, t_1] oder Zeitvektor `t0:dt:t1`

`y0` Anfangswert $y_0=y(t_0)$

```
function dy = funktionsname(t,y)
```

`t` ein Skalar `y` ein Spaltenvektor

`dy` ein Spaltenvektor mit gleicher Länge wie `y`

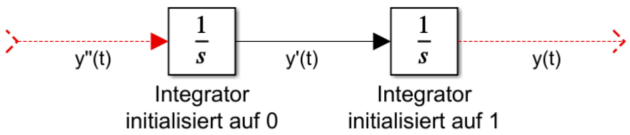
```
function [dy] = dlg1(t,y)
        dy = 2*t*y;

[t, y] = ode23(@dlg1, [-1,1], exp(1));
Plot(t,y) ;
```

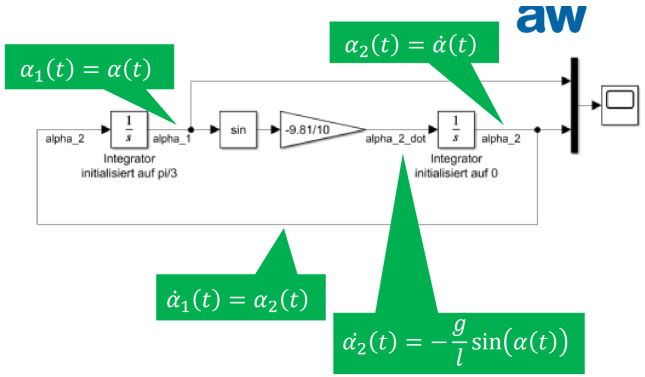
```
function [alphadot] = pendgl(t,alpha)
        l = 10                          % Pendellänge
        g = 9.81                        % Erdbeschleunigung
        alphadot = [0;0]                % Initialisierung
        % Darstellung der Differentialgleichung
        alphadot(1) = alpha(2);
        alphadot(2) = -(g/l)*sin(alpha(1);

[t, loesung] = ode23(@pendgl, [0,25], [pi/3,0]);
plot(t,loesung(:,1),'r-',t,loesung(:,2), 'g-');
```

Differenzialgleichungen mit Simulink

1. Formen Sie die DGL so um, dass die **Ableitung** nach der **höchsten Ordnung alleine links** der Gleichung steht
2. Starten Sie beim Modellieren von Differentialgleichungen im Simulink mit den Integratoren.

$$\ddot{y}(t) + \dot{y}(t) = 0 \rightarrow \ddot{y}(t) = -\dot{y}(t)$$





Simscape / Signalfluss

`simscape` Simscape starten

`PS-Simulink Converter` For Scope

Stateflow

`sfnew` Stateflow starten

Codegenerierung

1. Funktion schreiben
2. Mittels Befehl «mcc» eine Windows Exe Datei erzeugen (compilieren)
   `mcc -m <skript.m>` erzeugt Standalone-Exe
3. myPlot.exe benötigt ein Argument beim Aufruf
4. Windowskonsole zum Verzeichnis des Programm navigieren
   - `cd` für change directory
5. `myPlot.exe 4` (Aufrufwert 4)

Automatisch Codegenerierung

- Apps können auch per Matlab / Simulink App erzeugt werden
  - Siehe Dokumentation «Create Standalone Application from MATLAB»
- Umfangreiche Einstellungen möglich