



# Kurze MatLab-Einführung

- 1 Kurze Übersicht zu MatLab**
- 2 Einführung in MatLab
- 3 Modellierung mit Simulink



## 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.

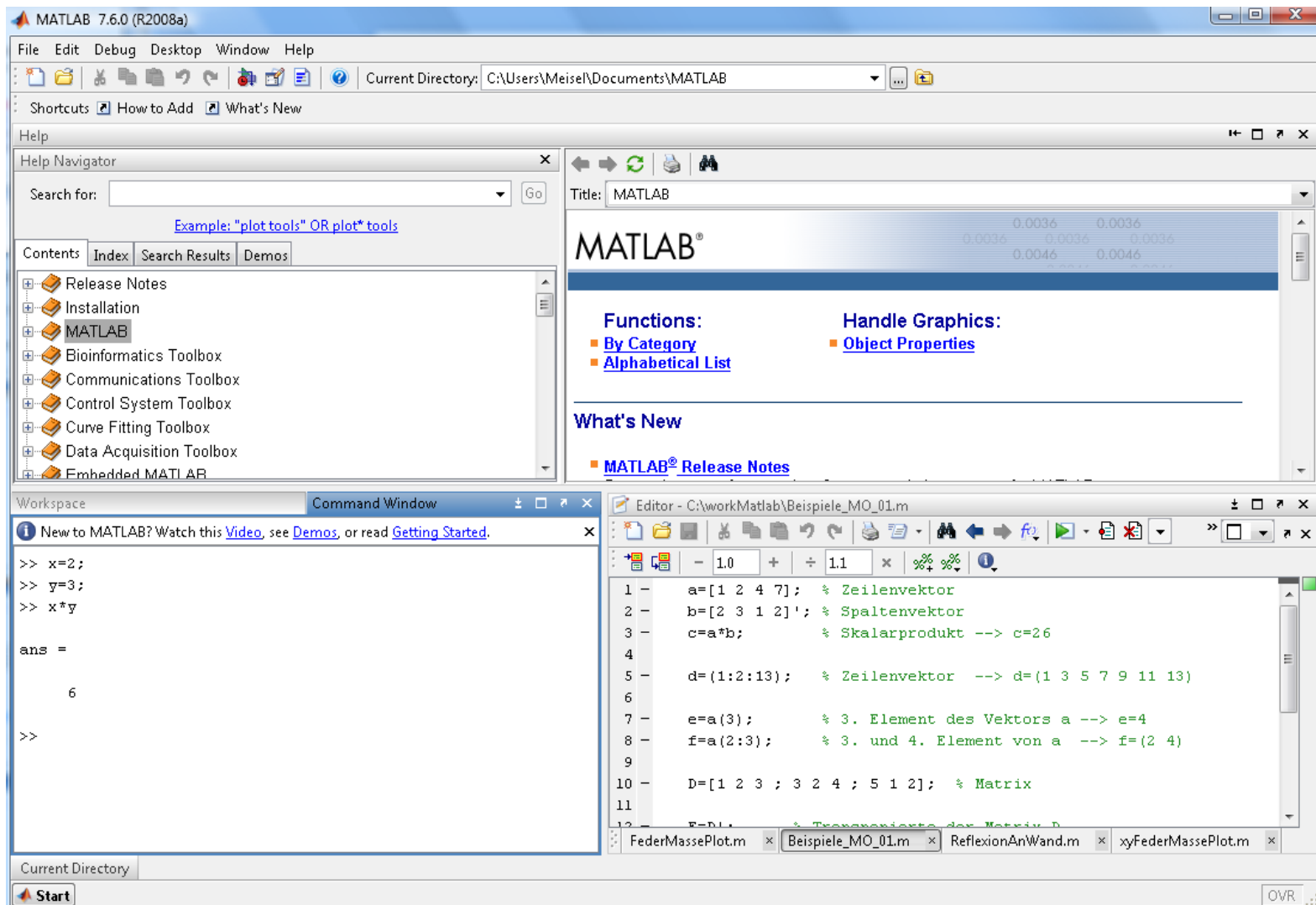
### Simulink

- grafisch interaktive Benutzeroberfläche zur Modellierung und Simulation dynamischer Systeme

### Stateflow

- grafisch interaktive Modellierung von Zustandsautomaten
- eingebunden in Simulink

## 1.2 Matlab – Desktop





## 1.2.1 Command Window

### Direkte Eingaben und Berechnungen

```
Workspace Command Window
i New to MATLAB? Watch this Video, see Demos, or read Getting Started.
>> Vec_a=[3 4];
>> Vec_b=Vec_a';
>> Vec_a*Vec_b

ans =

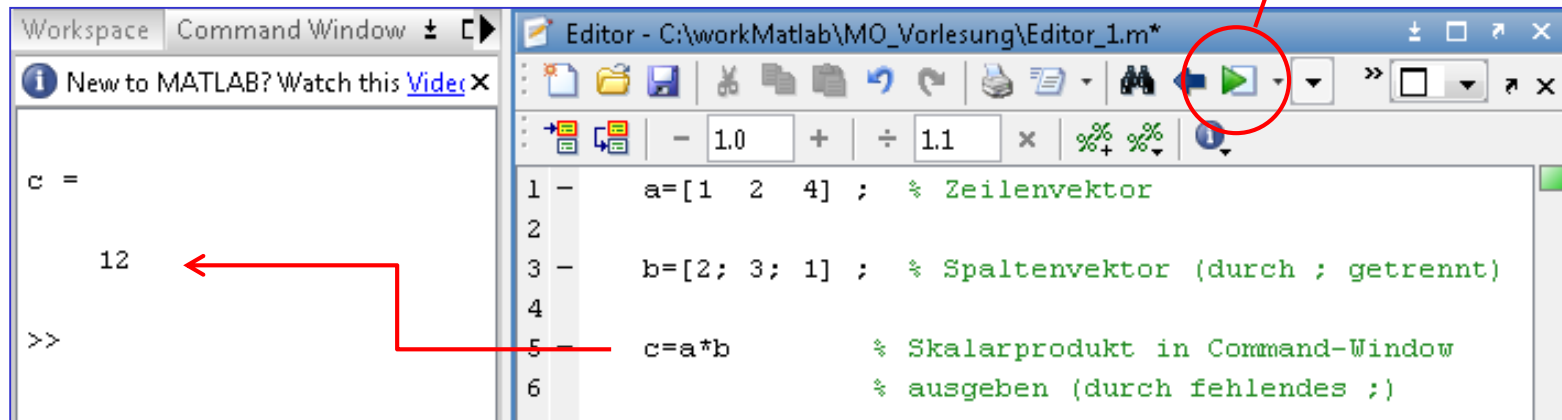
    25

>> |
```

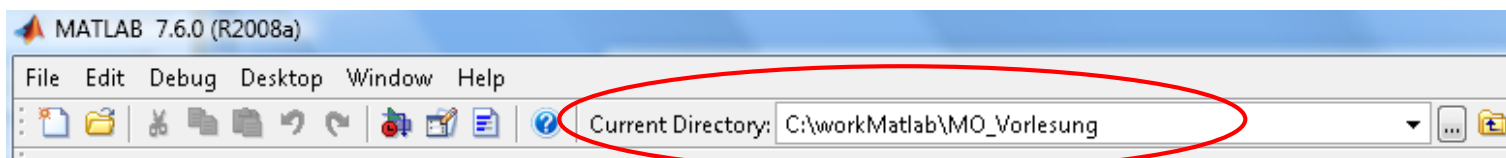
- Vec\_a ist ein Zeilenvektor
- durch ; wird Ausgabe unterdrückt
- Vec\_b ist die Transponierte von Vec\_a (durch ')
- Skalarprodukt berechnen und ausgeben (da ; fehlt)

## 1.2.2 Editor

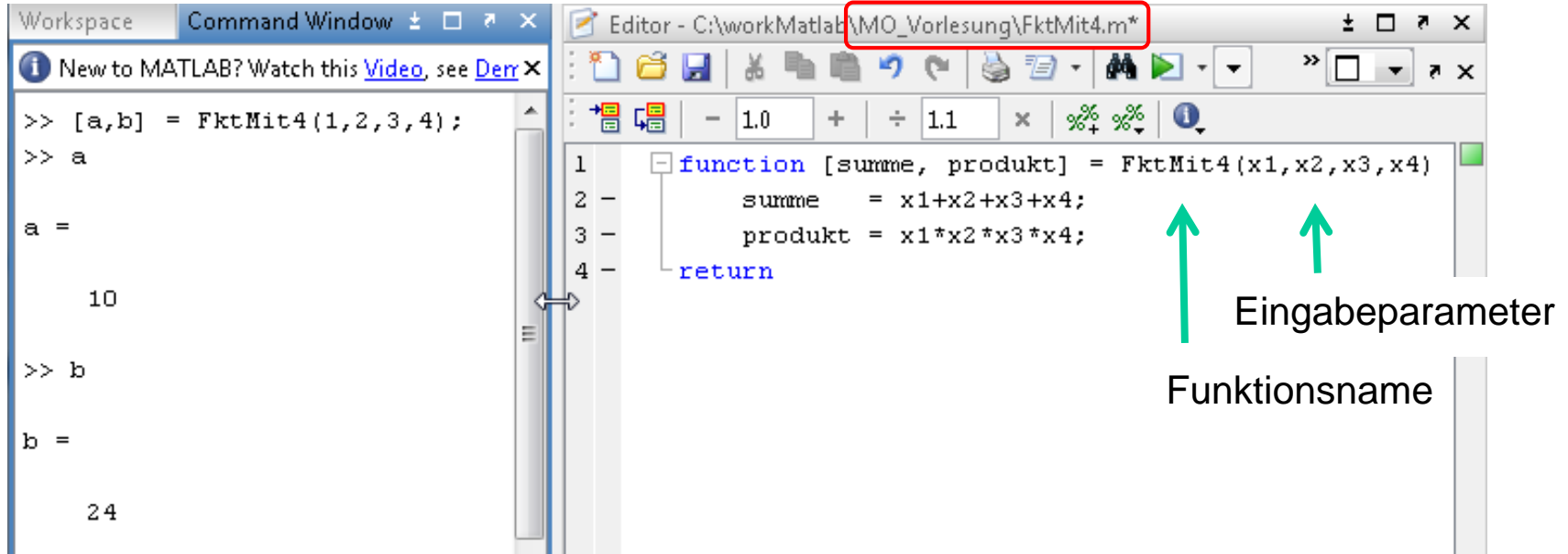
### Eingeben und starten von Skripten



- 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



```
Workspace Command Window Editor - C:\workMatlab\MO_Vorlesung\FktMit4.m*

New to MATLAB? Watch this Video, see Der...

>> [a,b] = FktMit4(1,2,3,4);
>> a

a =

    10

>> b

b =

    24

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

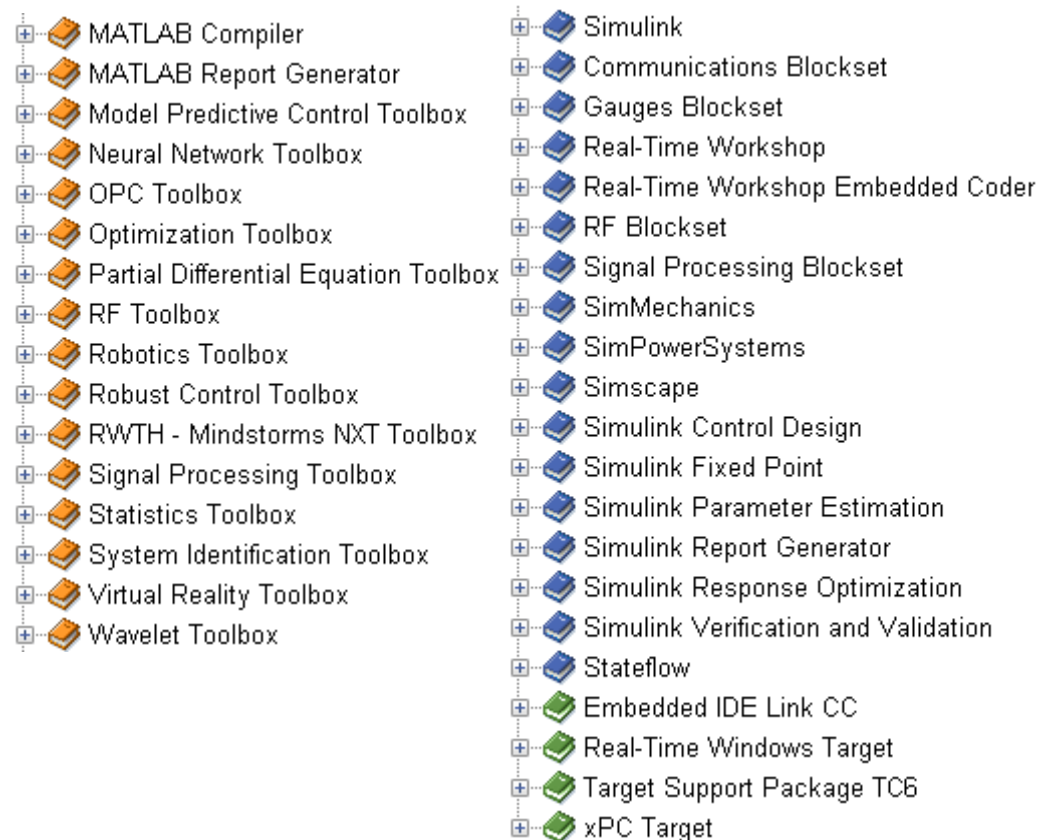
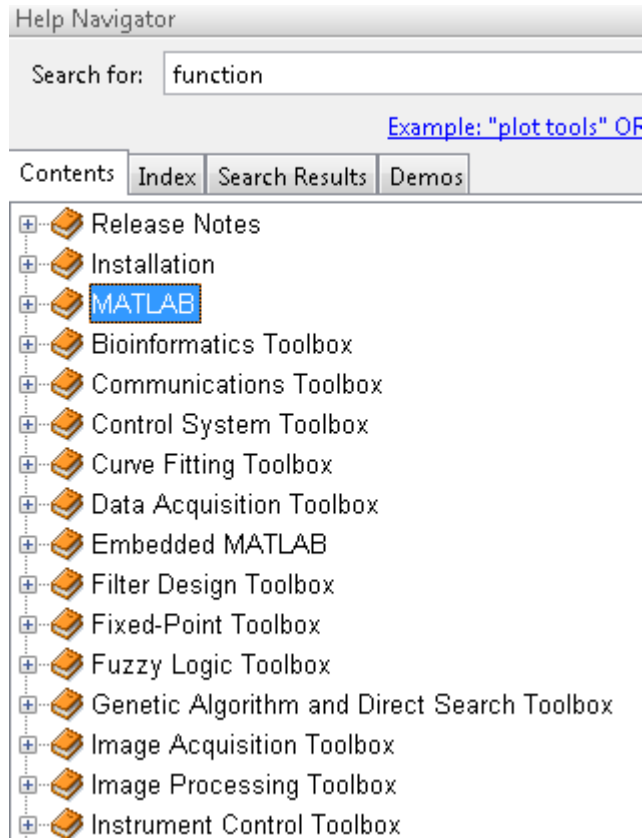
Eingabeparameter

Funktionsname

- **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.



## 1.3 Matlab – Toolboxes





## 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
```





# Kurze MatLab-Einführung

- 1 Kurze Übersicht zu MatLab
- 2 Einführung in MatLab**
- 3 Modellierung mit Simulink



## 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))`



## 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:  $A_7 = [1 \ 2 \ 3 \ ; \ 4 \ 5 \ 6 \ ; \ 7 \ 8 \ 9] ;$

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

### 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!
```

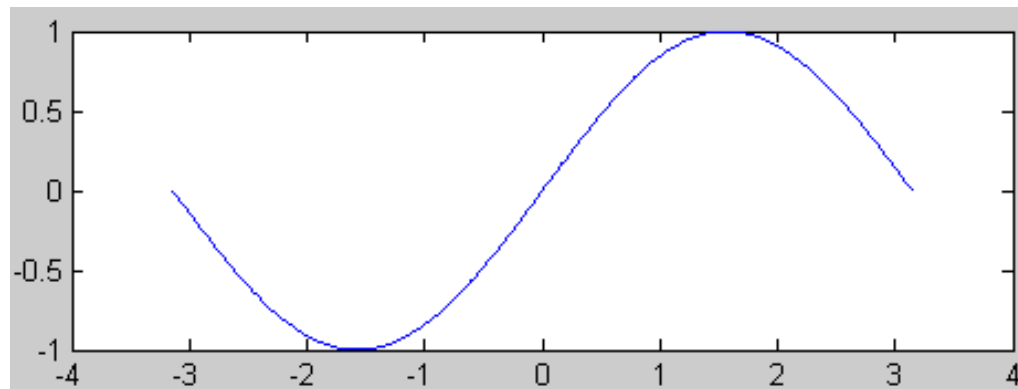
```
A17 = reshape(A1, 1, 6); % Erzeuge aus der 2x3 Matrix A1 eine
                          % 1x6-Matrix (A17 = [1 4 2 5 3 6])
                          % Anm.: Spaltenweise Umsortierung !
A18 = A1(end:-1:1 , :); % Zeilenreihenfolge umkehren
A19 = A1(: , end:-1:1); % Spaltenreihenfolge umkehren
```

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

## 2.3 Grafische Ausgaben (*sehr kleine Auswahl*)

### 2.3.1 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
```



Alternativen: Logarithmische Plots mit *semilogx()* , *semilogy()* , *loglog()*

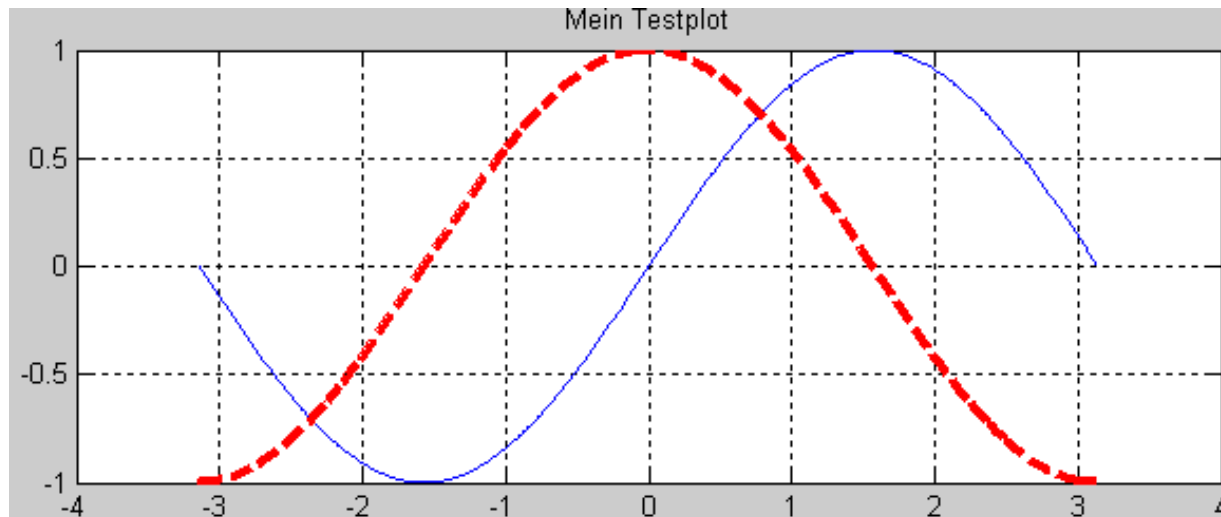


```
x = -pi:0.01:pi;    % Vektor von -Pi bis + Pi in 0.01-er Schritten
y1 = sin(x);        % Vektor der Sinuswerte
plot(x,y1);         % Funktion plotten

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
```

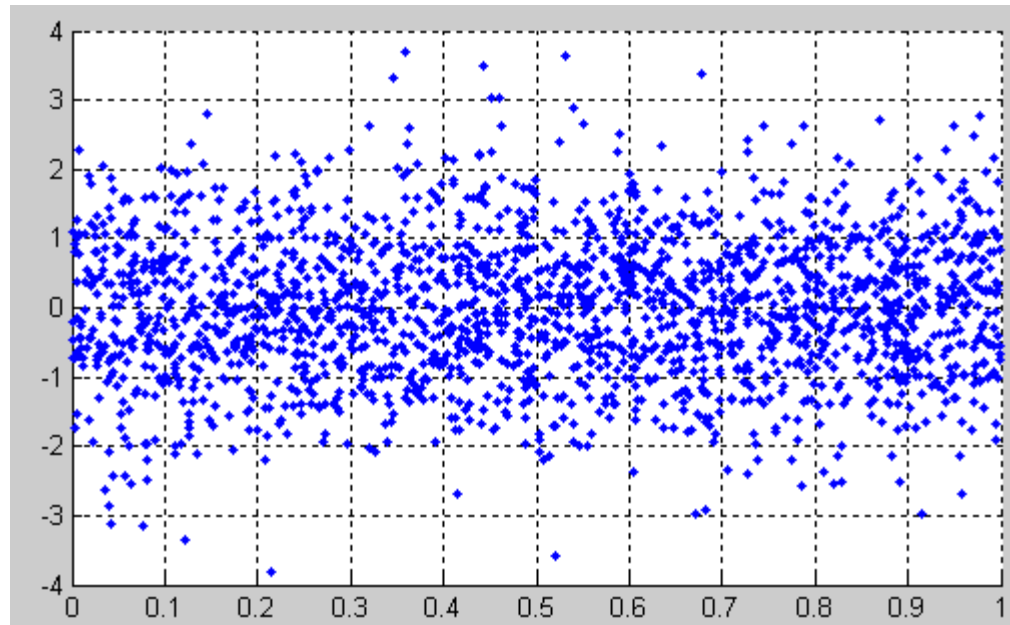




Symbol	Farbe	Symbol	Marker	Symbol	Linienart
b	blau	.	Punkt	-	durchgezogen
g	grün	o	Kreis	:	punktiert
r	rot	x	Kreuz	-.	strich-punktiert
c	cyan	+	Plus	--	strichliert
m	magenta	*	Stern		
y	gelb	s	Quadrat		
k	schwarz	d	Diamant		
w	weiß	v	Dreieck (unten)		
		^	Dreieck (oben)		
		<	Dreieck (links)		
		>	Dreieck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

## 2.3.2 scatter()

```
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
```



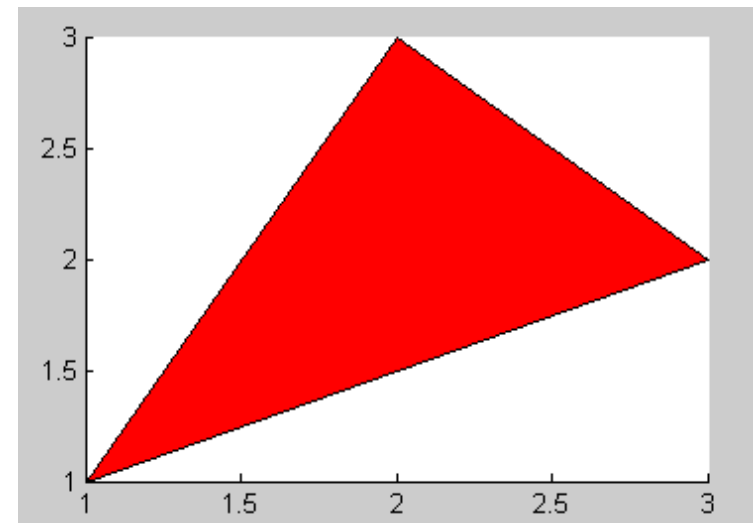
### 2.3.3 patch()

→ 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]



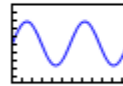


## 2.3.4 Kurze Übersicht

nicht vollständig

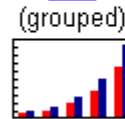
**Line  
Graphs**

[plot](#)



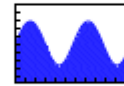
**Bar  
Graphs**

[bar](#)



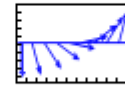
**Area Graphs**

[area](#)



**Direction  
Graphs**

[feather](#)



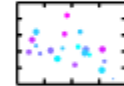
**Radial  
Graphs**

[polar](#)

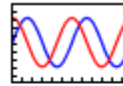


**Scatter  
Graphs**

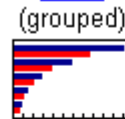
[scatter](#)



[plotyy](#)



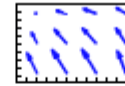
[barh](#)



[pie](#)



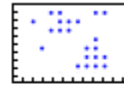
[quiver](#)



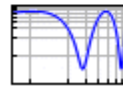
[rose](#)



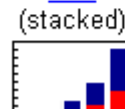
[spy](#)



[loglog](#)



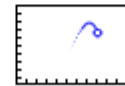
[bar](#)



[fill](#)



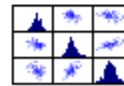
[comet](#)



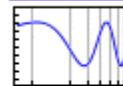
[compass](#)



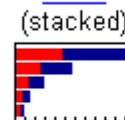
[plotmatrix](#)



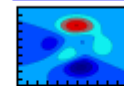
[semilogx](#)



[barh](#)



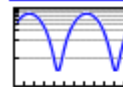
[contourf](#)



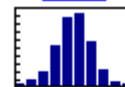
[ezpolar](#)



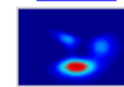
[semilogy](#)



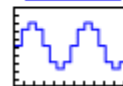
[hist](#)



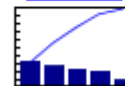
[image](#)



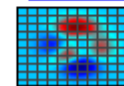
[stairs](#)



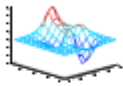
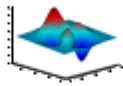
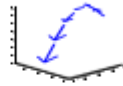
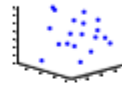
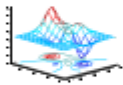
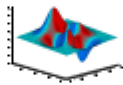
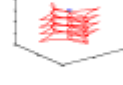
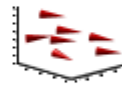
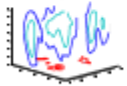
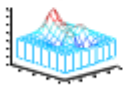
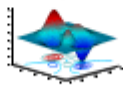
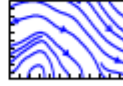
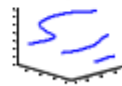
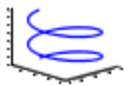
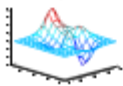
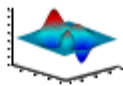
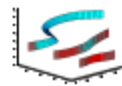
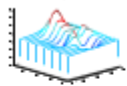
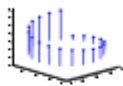
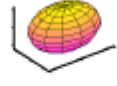
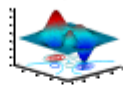
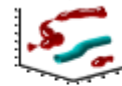
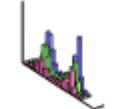
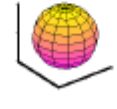
[pareto](#)



[pcolor](#)





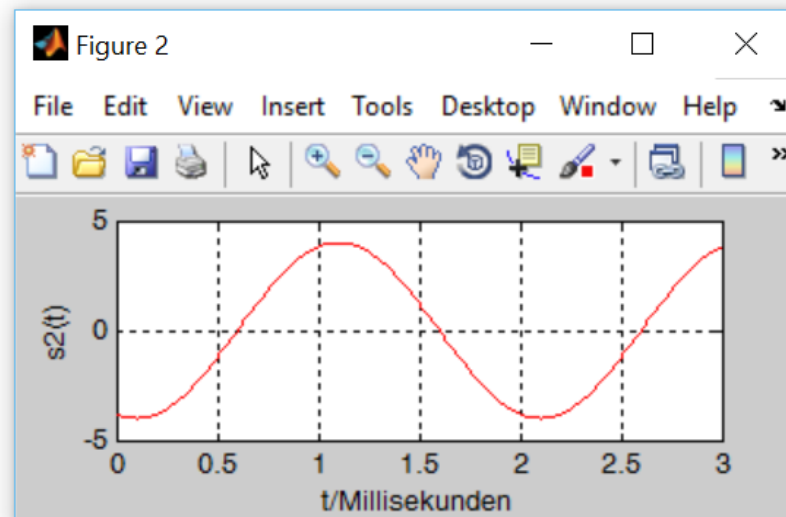
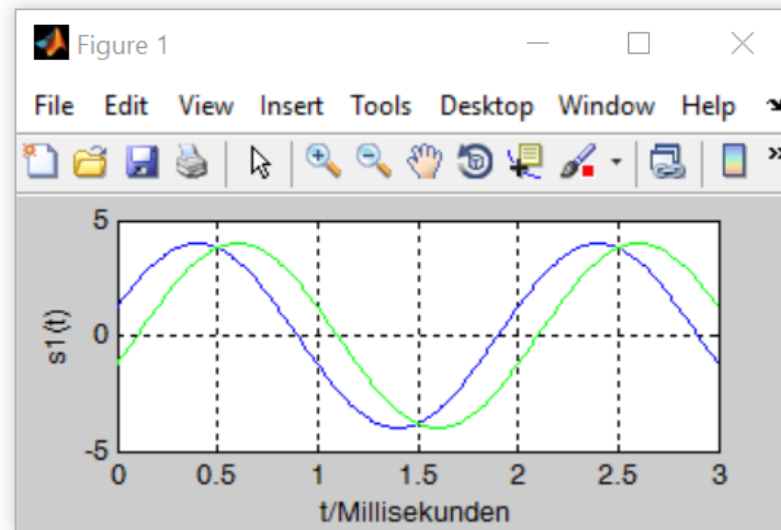
**Line Graphs****Mesh  
Graphs  
and Bar  
Graphs****Area  
Graphs and  
Constructive  
Objects****Surface  
Graphs****Direction  
Graphs****Volumetric  
Graphs**[plot3](#)[mesh](#)[pie3](#)[surf](#)[quiver3](#)[scatter3](#)[contour3](#)[meshc](#)[fill3](#)[surf1](#)[comet3](#)[coneplot](#)[contourslice](#)[meshz](#)[patch](#)[surfz](#)[streamslice](#)[streamlin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

## 2.3.5 Mehrere Ausgabefenster `figure( )`;

```
% Fenster 1
figure(1);
plot(t, y1);
grid on;
hold on;
% Achsenbeschriftung
xlabel('t/Millisekunden');
ylabel('s1(t)');

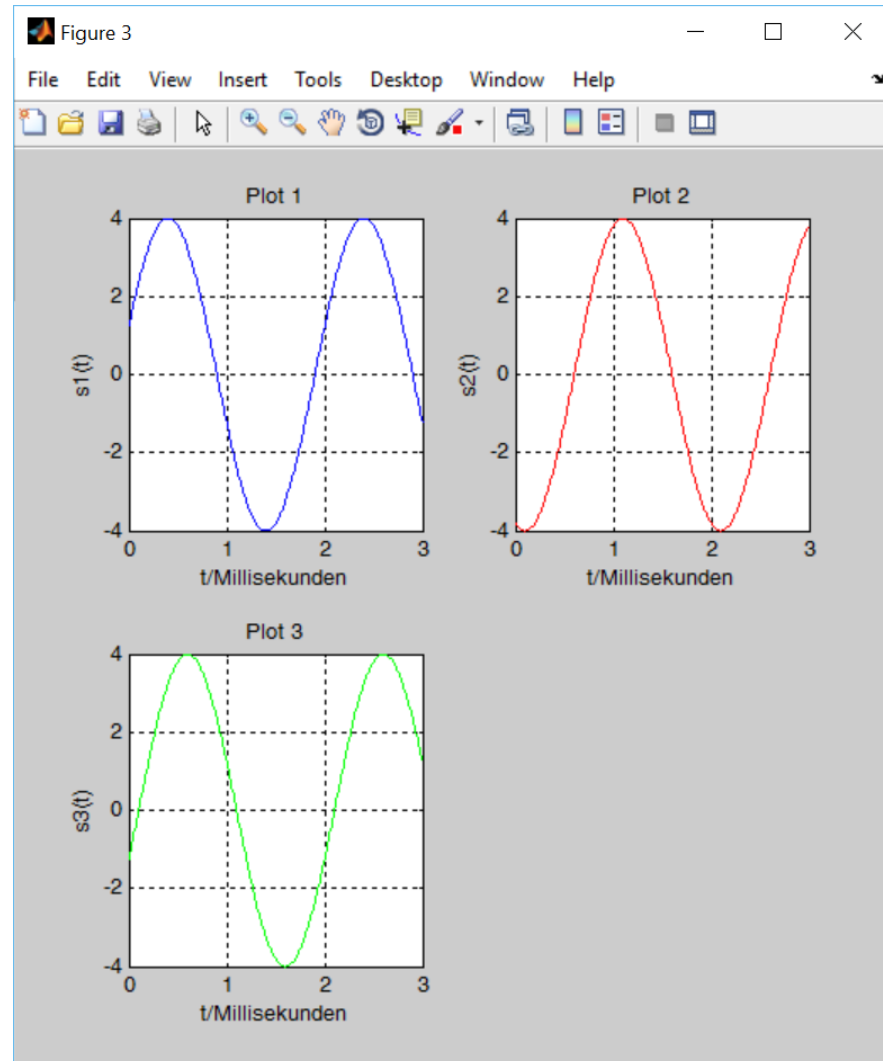
% Fenster 2
figure(2);
plot(t, y2, 'red');
grid on;
% Achsenbeschriftung
xlabel('t/Millisekunden');
ylabel('s2(t)');

% ... und wieder Fenster 1
figure(1);
plot(t, y3, 'green');
grid on;
```



## 2.3.6 Subplots

```
figure(3);  
subplot(2,2,1),  
plot(t, y1);  
title('Plot 1');  
grid on; hold on;  
% Achsenbeschriftung  
xlabel('t/Millisekunden');  
ylabel('s1(t)');  
  
subplot(2,2,2),  
plot(t, y2, 'red');  
title('Plot 2');  
grid on;  
% Achsenbeschriftung  
xlabel('t/Millisekunden');  
ylabel('s2(t)');  
  
subplot(2,2,3),  
plot(t, y3, 'green');  
title('Plot 3');  
grid on;  
% Achsenbeschriftung  
xlabel('t/Millisekunden');  
ylabel('s3(t)');
```





## 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
```



## 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
```



## 2.6 Grundfunktionen der Audio-Signalverarbeitung

```
% Wavedatei lesen --> y
[y, Samplefrequenz] = audioread('GitRiff.wav');
% Anzahl der Samples bestimmen
FileSize = size(y, 1);

% Platz für Ergebnisvektor allokieren (falls notwendig)
% yDist = zeros(FileSize, 1);

a = 30.0; % Verzerrungsparameter
% Verzerrung (distortion) des Signals mit Tangens-Hyperbolicus
yDist = 0.1*tanh(a*y);

% Ergebnissignal in Wavedatei schreiben
audiowrite('GitRiff_distorted.wav', yDist, Samplefrequenz);

% Ergebnissignal anhören
sound(yDist, Samplefrequenz);
```