Einführung in moderne Simulationstechniken
Kursteil: MATLAB / Simulink

Carsten Herzog

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

Kontakt

Carsten Herzog

RWTH Aachen: M.Sc. Techn. Informatik (Akustik und dig. Signalverarb.)
BMW: Doktorand (Schwingungskomfort für autonome Fahrzeuge)

Institut für Steuer- und Regelungstechnik
http://www.unibw.de/lrt15

Gebäude 41/300, Raum 2313
Tel.: +49 151 601 56797
E-Mail: carsten.herzog@bmw.de

Terminvereinbarung per E-Mail

Kursteil: MATLAB / Simulink

- **Inhalt:**
  - Teil 1: Einführung in MATLAB
  - Teil 2: Einführung in Simulink
  - Übungsaufgaben

- **Literatur:**
  - Angermann, Beuschel, Rau, Wolfrath, *MATLAB – Simulink – Stateflow*, Oldenburg Verlag, München, 6. Auflage 2009. http://www.matlabbuch.de
  - Stein, *Einstieg in das Programmieren mit MATLAB,* Hanser Verlag, 3. Auflage 2011. http://www.hanser.de
  - weitere Bücher in der Uni-Bibliothek oder im Buchhandel

- **Online:** www.mathworks.com

Teil 1: Einführung in MATLAB

- Was ist MATLAB?
- MATLAB Standardansicht
- MATLAB als Taschenrechner
- MATLAB Programmierung
- Anwendung in der Regelungstechnik

Teil 2: Einführung in Simulink

- Was ist Simulink?
- Vorgehen und Bibliotheken
- Beispiele und Aufgaben
- Subsysteme
- MATLAB Funktionen in Simulink

Übungsaufgaben

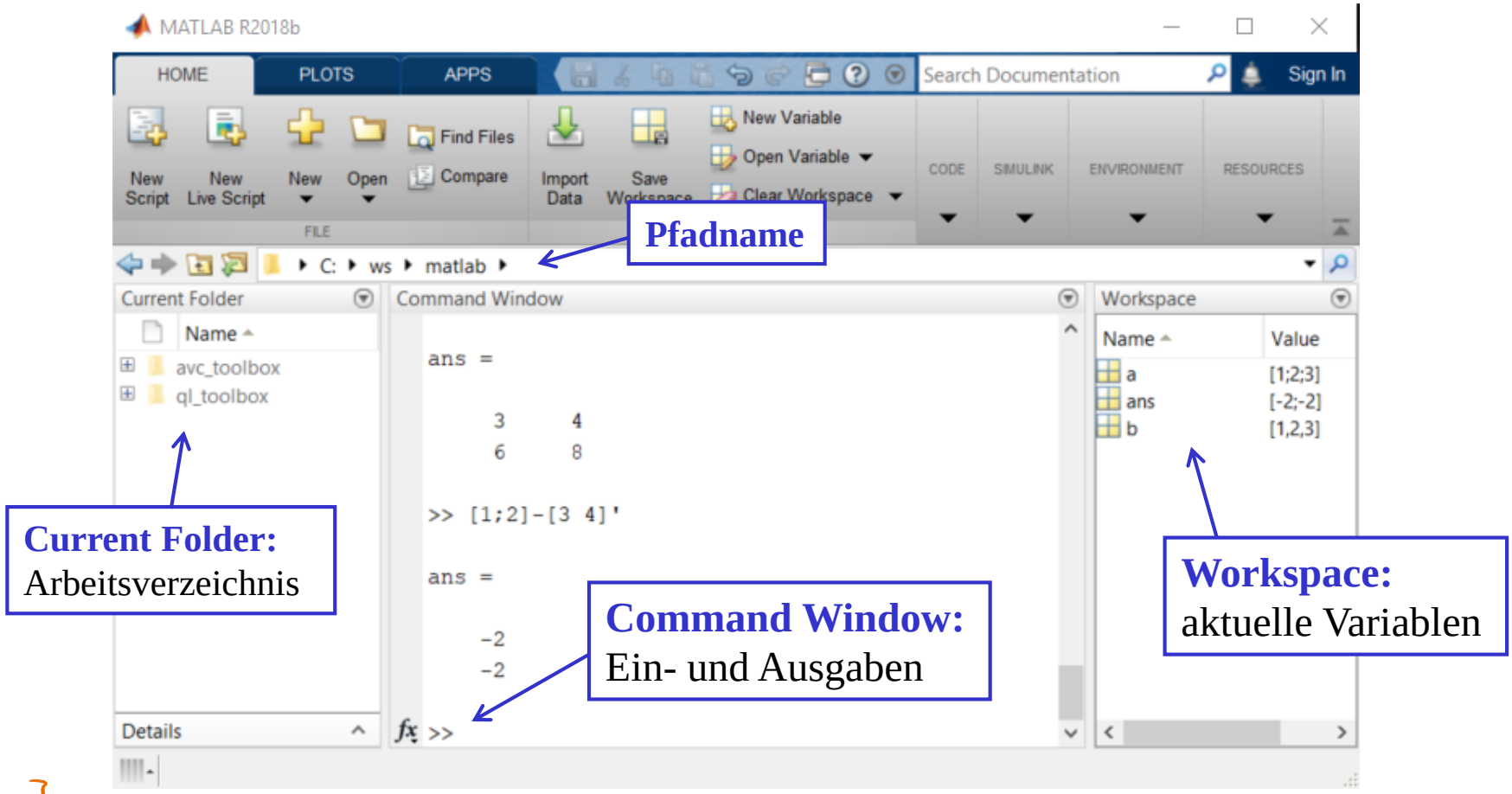
Was ist MATLAB?

- Programmiersprache und -umgebung für wissenschaftlich–technische Berechnungen
- Hersteller: *The MathWorks, Inc.*, Natick, MA, USA (gegr. 1984)
- Entstanden in den 1970er Jahren, auf Grundlage von Fortran77- Softwarepaketen aus dem Bereich der Numerischen Linearen Algebra.
- Basisdatentyp: Matrix → „**Mat**rix **lab**oratory"
- MATLAB = Basismodul + Toolboxes

  Basismodul:
  - Kontroll- und Datenstrukturen (Syntax an C angelehnt),
  - I/O einschl. Schnittstellen zu externer Software (Fortran, C, Java) und Hardware,
  - mathematische Funktionen,
  - 2D-, 3D-Graphik, ...

  Toolboxes:
  - Simulink,
  - Control System Toolbox,
  - Optimization Toolbox,
  - ...

→ MATLAB 2018b öffnen



MATLAB als Taschenrechner

- **Operationen:**

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

- **Konstanten:**

`pi`	Kreiszahl
`exp`	e-Funktion (Euler‘sche Zahl)
`inf`	Unendlich
…	

- **Variablen:** Zuweisung an beliebige Variablen möglich:

```
>> a = 2; b = 3.71;
>> a + b;
```

  - Ergebnisvariable `ans`
  - Variablen sind im aktuellen workspace definiert
  - Namenskonflikte mit Konstanten und Funktionen vermeiden

```
>> var = 5; V = var(a); % => ERROR
```

MATLAB als Taschenrechner

```
>> 2+3^2 % + 3     % alles nach '%' ist Kommentar
ans = 11           % Ergebnis in Variable 'ans' => siehe workspace

>> ans * ans;      % Befehl mit ';' beenden => keine Ausgabe
                   % Achtung: 'ans' wird überschrieben!

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

>> X1 = x1 + 1i    % 1i, 1j: imaginäre Einheit
X1 = 1.0000 + 1.0000i
```

MATLAB als Taschenrechner

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

>> 1x = 5           % Bezeichner beginnen mit Buchstaben
   1x = 5
    ↑
Error: Invalid expression. Check for missing ...

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

MATLAB als Taschenrechner: Vektoren

```
>> v = [1 2 3];    % Zeilenvektor: Leerzeichen bzw. ','
>> v = [1;2;3]     % Spaltenvektor: ';'
v =  1             % alternativ mit Transposition: >> v'
     2
     3

>> [1:3]           % Zeilenvektor: [startVal:stepSize:endVal]
ans = 1  2  3
>> [1:0.5:3]       % Angabe der Schrittweite
ans = 1.0000  1.5000  2.0000  2.5000  3.0000

>> ans(2:4)        % Zugriff auf Komponenten eines Vektors
ans = 1.5000  2.0000  2.5000
>> ans(1)
ans = 1.5000
```

MATLAB als Taschenrechner: Vektoren

```
>> [1;2] + [3 4]          % Dimensionen: 2x1 + 1x2
ans = 4   5               % Ergebnis: 2x2
      5   6

>> [1;2] * [3 4]          % Dimensionen: 2x1 * 1x2
ans = 3   4               % Ergebnis: 2x2
      6   8

>> [1;2]' * [3 4]         % Dimensionen: 1x2 * 1x2
Error using  *
Incorrect dimensions for matrix multiplication. Check ...

>> size(ans)
ans = 2   2
```

MATLAB als Taschenrechner: Vektoren

Hinweis: `>> doc % MATLAB Hilfe`

Aufgabe:

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

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

- Lasse dir jeweils das zweite, dritte und beide Elemente von `a` und `b` ausgeben

- Welche der folgenden Rechenoperationen lassen sich ausführen: `a*b;a'*b;a+b;a'+b`

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

MATLAB als Taschenrechner: Vektoren

Lösung:

```
>> size(a)
ans = 4     1      % gibt Dimension des Vektors an
                   % Spaltenvektor entspricht Nx1-Matrix
>> length(b)
ans = 4            % größte Matrix-Dimension

>> max(a)
ans = 8            % größter Eintrag im Vektor
                   % >> min(b): kleinster Eintrag im Vektor
>> mean(a)
ans = 5            % arithmetischer Mittelwert der Einträge

>> sum(b)
ans = 21           % Summe der Einträge
```

MATLAB als Taschenrechner: Vektoren

Lösung:

```
>> a(2)
ans = 1          % zweites Element in a; >> a(2,1)

>> b(3)
ans = 8          % drittes Element in b; >> b(1,3)

>> a(2:3)
ans = 1          % bei Auswahl mehrerer Elemente bleibt
      7          % die Art des Vektors erhalten
                        % (Spalten- bzw. Zeilenvektor)
```

- `a*b, a+b` und `a'+b` lassen sich ausführen
- `a'*b` lässt sich nicht ausführen, da Dimensionen nicht passen: 1x4 * 1x4

- Bei der Eingabe von zwei oder mehr Vektoren geben die Funktionen `min()` und `max()` einen neuen Vektor mit den kleinsten bzw. größten Einträgen aus.

MATLAB als Taschenrechner: Matrizen

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

>> m(2,3)                 % Zugriff auf Komponenten einer Matrix
ans = 3                   % m(Zeile,Spalte)
>> m(3,2)
ans = 7

>> m(1:3,2:3)*[1;2]       % Multiplikation: 3x2 * 2x1 = 3x1
ans =  0
       8
      25
```

MATLAB als Taschenrechner: Matrizen

Aufgabe:

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

- Welche der in der vorherigen Aufgabe benutzten Befehle lassen sich auch auf Matrizen anwenden?
  `size(), length(), max(), min(), mean(), sum()`
- Was wird berechnet?
- Greife auf die Elemente der 1. Spalte von A und B einzeln zu.
- Lasse dir die Elemente auf der Hauptdiagonalen von A und B ausgeben.
- Wende zusätzlich die folgenden Befehle auf beide Matrizen an und versuche deren Funktionsweise zu ermitteln:
  `diag(), trace(), eig(), inv()`
- Berechne $A + B$, $A * B$, $A.* B$, A/B und $A./B$.
- Wie unterscheiden sich $A * B$ und $A.* B$ bzw. A/B und $A./B$?
- Was bewirken die Funktionen `zeros(n)`, `ones(n)` und `eye(n)` mit $n \in \mathbb{N}$?

MATLAB als Taschenrechner: Matrizen

Lösung:

```
>> size(A)
ans = 3     3                % Dimensionen der Matrix
>> length(B)
ans = 3                      % größte Dimension der Matrix

>> max(A)
ans = 5  4  2                % größter Eintrag der Spaltenvektoren
>> min(B,[],2)
ans = 1                      % kleinster Eintrag der Zeilenvektoren
      2
      1

>> mean(A)
ans = 3.667  3.333  1.33      % Mittelwert der Spaltenvektoren
                              % >> mean(A,2)
```

MATLAB als Taschenrechner: Matrizen

Lösung:

```
% Elemente der ersten Spalte
>> [A(1,1); A(2,1); A(3,1)]      % ans = [4;5;2]

% Elemente der Hauptdiagonale
>> [B(1,1); B(2,2); B(3,3)]      % ans = [1;2;3]
>> diag(A)
ans = [4;3;1]                    % Elemente der Hauptdiagonale
>> trace(B)
ans = 6                          % Spur der Matrix (Summe der
                                 % Elemente der Hauptdiagonale)
>> eig(A)
ans =  8.6617 + 0.0000i          % Spaltenvektor mit Eigenwerten
      -0.3309 + 1.2737i
      -0.3309 - 1.2737i
>> inv(B)                  % Inverse der Matrix (sofern möglich)
```

MATLAB als Taschenrechner: Matrizen

Lösung:

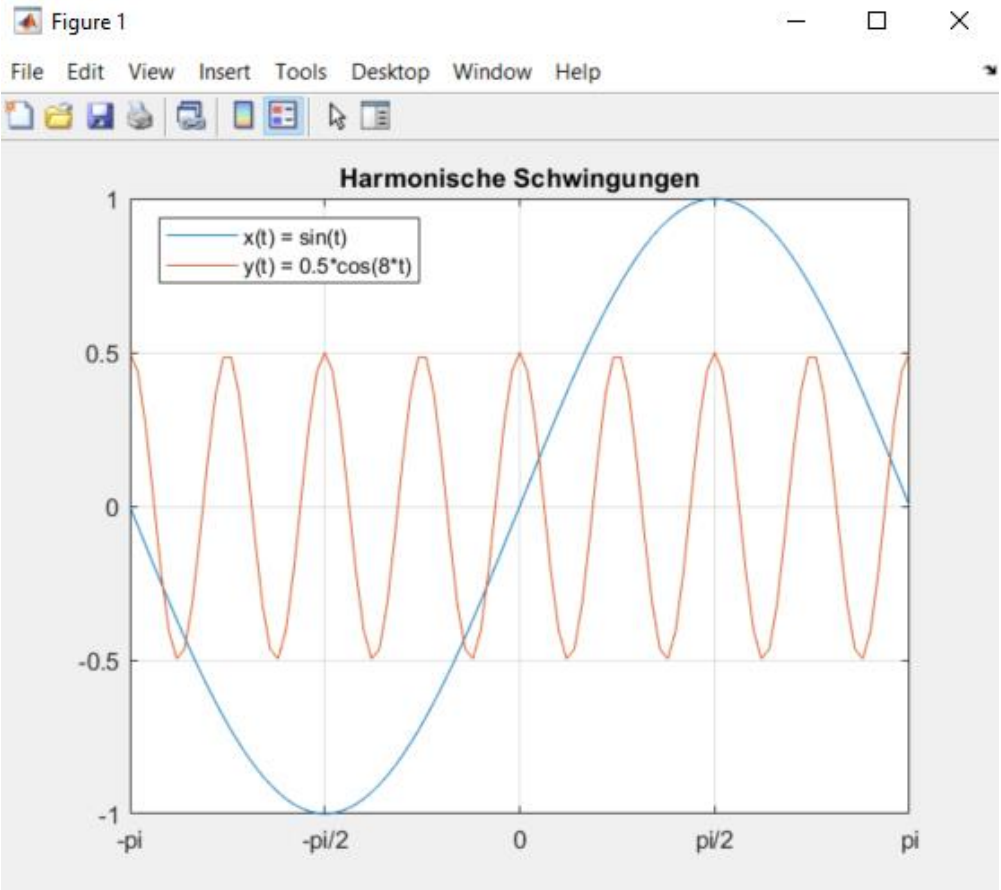
```
>> A*B
ans = 17    12    38              % normale Matrix-Multiplikation
      16    12    40
      16    11    29
>> A.*B
ans = 4     3    10              % elementweise Multiplikation
     15     6     4
      4     4     3
>> A/B
ans = -0.7500    2.7500   -1.7500        % entspricht A*inv(B)
      -1.5000    2.5000   -0.5000        % auch elementweise möglich
       0.2500    5.7500   -7.7500        % >> A./B
>> zeros(n); ones(n); eye(n);
   % Matrizen der Dimension n mit
   % Nullen und Einsen bzw. Einheitsmatrix
```

MATLAB als Taschenrechner: 2D-Grafiken

```
>> figure                        % neues Grafik-Fenster
>> t = [-pi:2*pi/100:pi];        % Vektor der Abtastzeitpunkte
>> x = sin(t);                   % Vektor der Funktionswerte x(t)
>> y = 0.5 * cos(8*t);           % Vektor der Funktionswerte y(t)
>> plot(t,x,t,y);                % x(t), y(t) darstellen
>> plot(t,x);                    % x(t) darstellen; y verschwindet
>> hold on                       % x(t) soll bleiben
>> plot(t,y,'g');                % y(t) kommt dazu, diesmal grün
>> grid on                       % Gitternetz darstellen

% optional: legend, xlabel, ylabel, title, ...
```

MATLAB als Taschenrechner: 2D-Grafiken




```
>> close              % Grafikfenster schließen
```

MATLAB Programmierung: Hilfe

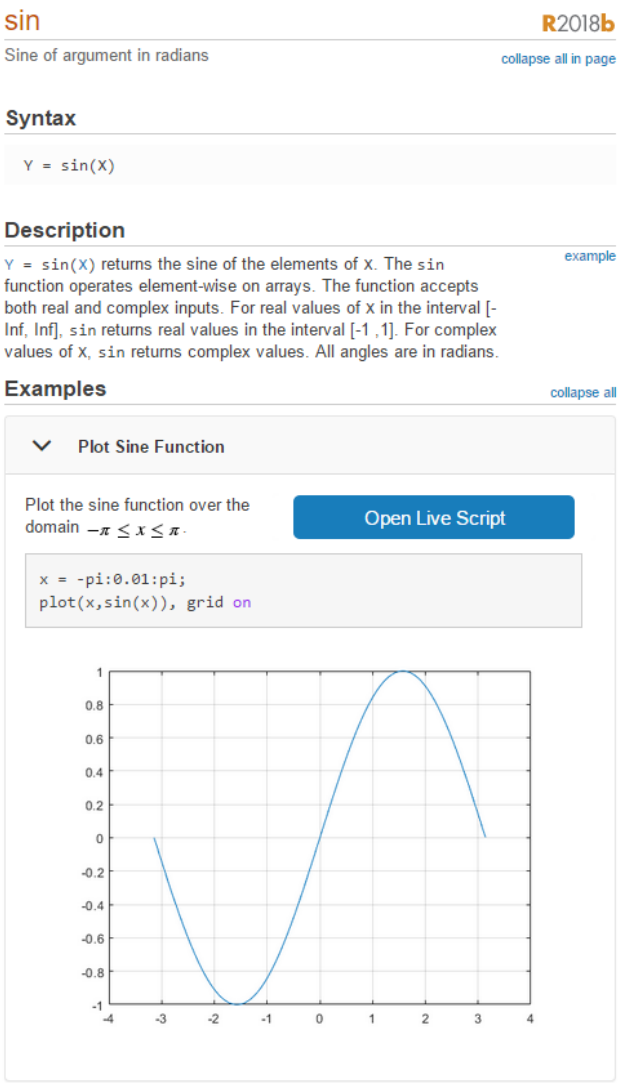
```
% Hilfe im Command Window
>> help sin
```

```
sin    Sine of argument in radians.
   sin(X) is the sine of the elements of X.

   See also asin, sind, sinpi.

   Reference page for sin
   Other functions named sin
```

```
% ausführliche Hilfe
% im Help Browser
>> doc sin
```



MATLAB Programmierung: Dateien schreiben und lesen

```
>> clear; clc;
>> mkdir meintest          % Anlegen eines Verzeichnisses
>> cd meintest             % als Arbeitsverzeichnis wählen
>> x=1; y=1; z=1;
>> save xydatei x y        % x, y => Datei xydatei.mat
>> save xyzdatei           % x, y, z => Datei xyzdatei.mat
>> x=2;y=2;z=2;
>> load xydatei
>> [x y z]                 % ans = 1   1   2
>> load xyzdatei
>> [x y z]                 % ans = 1   1   1
>> cd ../                  % Arbeitsverzeichnis wechseln
>> rmdir meintest s        % Arbeitsverzeichnis löschen
                           % inkl. Daten
```

MATLAB Programmierung: Schleifen

for-Schleife:

```
for <index> = <initVal>:<endVal>          % Anweisungen werden
    <statements>                          % solange ausgeführt,
end                                       % bis index größer als
                                          % endVal ist
```

Beispiel:

```
N = 4;
for i = 1:N
    a(i) = exp(i*1i*pi/2);     % Anweisung wird N-mal ausgeführt
end
```

- Schleifenaufruf und -ende benötigen kein Semikolon
- Anweisungen im Schleifen-Rumpf sollten mit Semikolon abgeschlossen werden um Ausgaben im Command Window zu vermeiden

MATLAB Programmierung: Schleifen

while-Schleife:

```
while <expression>          % Anweisungen werden solange ausgeführt,
    <statements>            % wie die Bedingung erfüllt ist
end
```

Beispiel:

```
i = 1;
N = 4;
while i <= N
   b(i) = exp(i*1i*pi/2);    % Anweisung wird N-mal ausgeführt
   i = i+1;
end
```

MATLAB Programmierung: Bedingungen

if-else-Bedingung:

```
if <expression>            % wenn Bedingung erfüllt ist,
   <statements>            % werden die Anweisungen ausgeführt
else
   <statements>
end
```

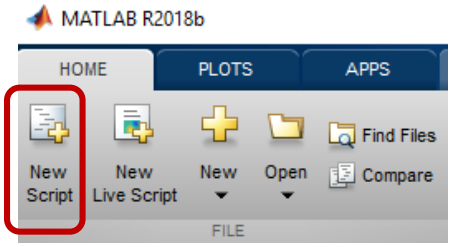
Beispiel:

```
i = 1;
if i >= 2
   disp('i >= 2')          % Ausgabe in Command Window
else
   disp('i < 2')
end
```

MATLAB Programmierung: Skripte

- Skripte (Programme) und Funktionsdefinitionen speichert MATLAB in Dateien vom Typ `<name>.m`
- Implementierung und Debugging im MATLAB-Editor

  ➔ New Script oder `>> edit`



- **Skripte:**
  - Abarbeitung so, als würden Anweisungen nacheinander ins Command Window eingegeben
  - Nutzung durch Aufruf des Dateinamens ohne Endung im Command Window
  - Neben Rechenoperationen (z. B. `+`, `*`) und Funktionen (z. B. `exp(x)`) auch Verwendung von anderen Anweisungen wie Schleifen (z. B. `for`) und Bedingungen (z. B. `if else`) möglich

Beispiel:

1. Skript erstellen

```
figure
hold on                          % Man teste auch 'off'.
t = 0:0.01:2*pi;
for k = 1:3
plot(t, k*sin(k*t));
pause(1);                        % Was bewirkt dieser Befehl?
end
```

2. Skript im Arbeitsverzeichnis speichern unter `myScr.m`

3. Programm ausführen: `>> myScr  % ohne Dateiendung`

MATLAB Programmierung: Funktionen

Funktionen:

```
function [y1,y2,...,yN] = myFnc(x1,x2,...,xM)
% Hilfetext
<statements>        % MATLAB-Anweisungen
y1 = ...            % Zuweisung der Funktionswerte
end                 % Ende der Definition von myFun
```

- Schlüsselworte: `function`, `end`
- Name der Funktion (`myFnc`) ist frei wählbar und muss mit dem Dateinamen übereinstimmen
- Namen der Input- und Outputparameter, hier z. B. `x1` bzw. `y1`, sind frei wählbar
- Variablen die im Funktionsblock definiert werden sind nur im lokalen workspace sichtbar
- Input- und Outputparameter können komplexe Datentypen sein (z. B. Matrix, Klasse, ...)
- Kommentar direkt nach der Funktionsdefinition wird als Hilfe angezeigt `>> doc myFnc`

MATLAB Programmierung: Funktionen

Beispiel:

1. Funktion erstellen

```
function [s,p] = myFnc(v)
% s: sum / p: product / v: vector
s = sum(v);
p = prod(v);
end
```

2. Funktion im Arbeitsverzeichnis speichern unter `myFnc.m`
3. Funktion in Command Window oder in Skript verwendbar

```
>> [s,p] = myFnc([1:10])          % s = 55 ; p = 3628800

>> doc myFnc                      % Help Browser
```

Aufgaben:

1. Erstelle ein Skript, dass die Funktion $f(x) = x^2 + 4 + e^x$ auf dem Intervall $x = [-5,5]$ mit einer Schrittweite von n $= 0{,}1$ berechnet und anschließend grafisch darstellt. Experimentiere anschließend mit der Schrittweite und beobachte das Ergebnis.
   Hinweis: e^x kann mit der Funktion `exp(x)` berechnet werden

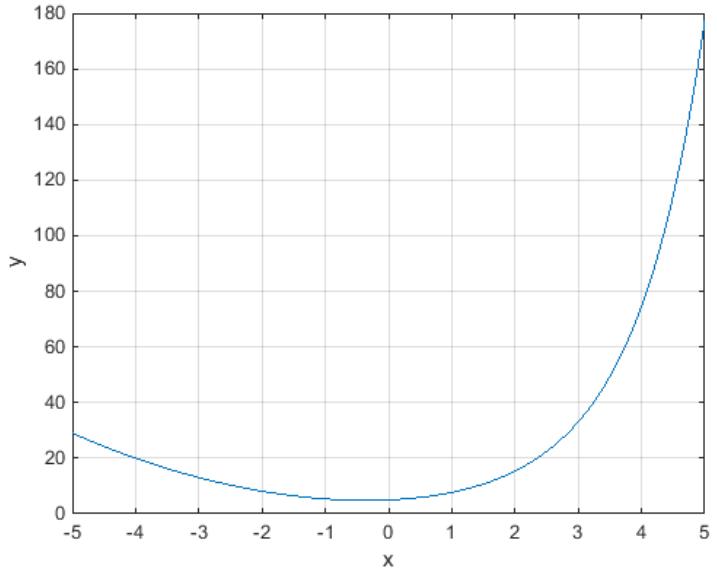
2. Implementiere eine Funktion, die den größten Wert auf der Hauptdiagonale einer beliebigen symmetrischen Matrix ausgibt.

3. Programmiere eine Funktion, die die Summe der Beträge aller Vektorelemente berechnet.
   Hinweis: Verwende eine `for`-Schleife und die Funktion `abs()`.

Lösung Aufgabe 1:

Erstelle ein Skript, dass die Funktion $f(x) = x^2 + 4 + e^x$ auf dem Intervall $x = [-5,5]$ mit einer Schrittweite von n = 0,1 berechnet und anschließend grafisch darstellt. Experimentiere anschließend mit der Schrittweite und schildere das Ergebnis.

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



Bei kleinerer Schrittweite wird der Plot genauer, bei größerer ungenauer.

Lösung Aufgabe 2:

Implementiere eine Funktion, die den größten Wert auf der Hauptdiagonale einer beliebigen symmetrischen Matrix ausgibt.

```
function maxVal = maxDiag(A)
% Funktion liefert den maximalen Wert der Hauptdiagonalen.
% A: Matrix
d = diag(A);
maxVal = max(d);
end
```

Test:

```
>> A = [1 2 ; 3 4]; maxDiag(A)          % ans = 4
```

Schachtelung von Funktionen möglich: `>> maxVal = max(diag(A));`

Lösung Aufgabe 3:

Programmiere eine Funktion, die die Summe der Beträge aller Vektorelemente berechnet.

```
function s = sumAbs(v)
% Funktion liefert die Summe der Beträge aller Vektorelemente.
% v: Spaltenvektor
s = 0;
N = size(v,1);                  % Anzahl der Elemente
for i = 1:N                     % Schrittweite 1 (default)
        s = s + abs(v(i,1));    % Summe der Beträge
end
end
```

Test:

```
>> v = [1;-2;3]; sumAbs(v)      % ans = 6
```