Einführung in Matlab – learning by doing

1. Variablen, Vektoren, Matrizen (siehe auch 1.5 und 1.6[1])

`a=7`	`a` wird als Skalar definiert (oder 1x1 Matrix)
`b=[1,2,3]`	`b` ist ein Zeilenvektor (oder 1x3 Matrix); nach Komma: neues Spalte
`c=[1 2 3]`	Leerzeichen haben die gleiche Bedeutung wie Kommata
`d=[1+2,a,4]`	
`f=[1;2;3]`	`f` ist ein Spaltenvektor (oder 3x1 Matrix); nach Semikolon: neue Zeile
`g=1:9`	Doppelpunkt Operator: <*Startwert*>:<*Endwert*> (Schrittweite standardmäßig 1)
`h=1:0.5:9`	Doppelpunkt Operator: <*Startwert*>:<*Schrittweite*>:<*Endwert*>
`i=1:2:9`	
`j=i(2)`	`j` wird als zweites Element von i gesetzt (Indizierung startet mit 1)
`k=i([1 3 5])`	`k` wird als Zeilenvektor (oder 1x3 Matrix) mit dem 1, 3 und 5 Eintrag von `i` definiert
`l=i(1:2:5)`	jetzt mit Kurzschreibweise (Doppelpunkt Operator)
`m=-1.23e4`	Exponentialdarstellung: $-1.23 \cdot 10^4$
`A=[1 2 3;` `4 5 6;7 8 9;10 11 12]`	`A` wird als 4x3 Matrix definiert; Matlab unterscheidet Groß-/Kleinschreibung
`disp(A)`	Ausgabe von `A`
`m=A(2,1)`	`m` ist Skalar mit Wert des Eintrages von A in der zweiten Zeile, erste Spalte
`B(3,4)=34`	`B` wird als 3x4 Matrix interpretiert; alle nichtbenannten Elemente werde Null gesetzt
`B(4,1)=41`	`B` wird zur 4x4 Matrix erweitert
`C=A(2,[1 3])`	`C` ist 1x2 Matrix mit Einträgen aus 2. Zeile und der 1. und 3. Spalte von `A`
`D=A(1:3,[1 3])`	`D` ist 3x2 Matrix von Einträgen der 1. bis 3. Zeile und der 1. und 3. Spalte von `A`
`size(A)`	Anzahl der Zeilen und Spalten von `A` als Zeilenvektor
`[n,m]=size(A)`	Anzahl der Zeilen von `A` in `n` und Anzahl der Spalten von `A` in `m`
`help size`	Hilfe zur Funktion `size`
`doc size`	Hilfe zur Funktion `size` im Hilfebrowser
`1j`	Komplexe Einheit
`z=-2+3j`	Definition komplexe Zahl
`X = [1j 2; z -1-j]`	Definition komplexwertiger Matrix
`who`	Ausgabe der definierten Variablen
`whos`	Ausgabe der definierten Variablen mit Zusatzinformationen
`clear`	löscht alle Variablen
`whos`	

[1] Die Angabe bezieht sich auf das Tutorial „Matlab – Eine Einführung“ von Christian Karpfinger und Boris von Loesch

2. Indizierung (siehe auch 1.6)

`clear`	
`a=1:12`	
`a(5:end)`	
`a(5:2:end-3)`	
`A=[1 2 3; 4 5 6; 7 8 9]`	
`A(:,1)`	gesamte erste Spalte
`A(end,[1 3])`	letzte Zeile, 1. und 3. Spalte
`A(:)`	Vektor aller Spalten der Matrix

3. Rechenoperationen (siehe auch 1.7)

`clear`	
`a=[1 2 3]`	1x3 Matrix
`b=a'`	Transponierte von a: 3x1 Matrix
`c=[10 11 12]`	1x3 Matrix
`d=[4;5;6]`	3x1 Matrix
`e=[14 15 16]'`	3x1 Matrix (durch Transponieren einer 1x3 Matrix)
`a+c`	
`a-c`	
`a-d`	Fehler: Dimensionen müssen übereinstimmen (R2017: führt auf 3x3 Matrix)
`a-d'`	
`a*d`	Matrixmultiplikation (entspricht hier Skalarprodukt)
`d*a`	3x1 Matrix mit 1x3 Matrix multipliziert ergibt 3x3 Matrix
`a*a'`	Quadratische Euklidsche Norm von a
`a.*c`	elementweise Multiplikation
`a.*d`	Fehler: Dimensionen müssen übereinstimmen (R2017: führt auf 3x3 Matrix)
`clear`	
`A=[1 2;3 4]`	
`B=[5 6; 7 8]`	
`x=[2 4]'`	
`b=[3 10]'`	
`A*x`	Matrix-Vektormultiplikation (entspricht 2x2 Matrix mit 2x1 Matrix multipliziert)
`A\b`	Lösen des Gleichungssystems A*x = b
`A*B`	Matrix-Multiplikation
`A.*B`	elementweise Multiplikation
`A^2`	entspricht A*A
`A.^2`	elementweise 2. Potenz

3. Elementare Funktionen (siehe auch 1.5 und 1.7.2)

`clear`	
`x=pi/2`	Konstante π
`sin(x)`	Analog: `cos`, `tan`, `exp`, `log`, `log2`, `log10`, `asin`, `acos`, `sqrt`
`xx=0:0.1:1`	Vektor von x-Werten
`sin(xx)`	Elementweise berechneter Vektor
`help elfun`	
`A = eye(3)`	3x3 Einheitsmatrix
`B = zeros(4,3)`	4x3 Matrix mit Einträgen 0
`C = ones(2,3)`	2x3 Matrix mit Einträgen 1

4. Graphiken/Ausgabe – Einführung (siehe auch 4.2)

`y=2:12;`	Semikolon am Schluss unterdückt die Ausgabe
`plot(y)`	Zeichne Werte von `y` über x-Werte von 1 bis Länge von `y`
`x=0:0.1:4`	
`y=sin(x)`	
`plot(x,y)`	Zeichne Werte von `y` über die Werte von `x` (beide Vektoren müssen gleich lang sein)
`xlabel('Zeit [s]')`	
`ylabel('Position [m]')`	
`title('Schwingung')`	
`xlim([0 pi])`	Legt den Bereich der x-Achse fest (analog: `ylim`)
`plot(x,y,'x')`	Verwende Kreuze zur Darstellung; weitere Optionen siehe `doc plot (LineSpec)`
`z=cos(x)`	
`plot(x,y,'-r',x,z,'-b')`	Darstellung zweier Kurven in einem Plot
`legend('Sin','Cos');`	Legende hinzufügen
`xx=0:4, yy=2.^xx`	
`semilogy(xx, yy, 'x-')`	Halblogarithmischer Plot (siehe auch `semilogx` und `loglog`)
`a=1; x=10.2;`	
`fprintf('%3i\n',a)`	Gib die Variable a als Ganzzahl (`%i`) mit Breite 3 aus und Zeilenvorschub (`\n`)
`fprintf('%8.3f\n',x)`	Gib die Variable x als Fixkommazahl (`%f`) mit Gesamtbreite 8 und 3 Nachkommastellen
`fprintf('%10.3e\n',x)`	Gib die Variable x in Exponentschreibweise (`%e`) mit Gesamtbreite 10 und 3 Nachkommastellen
`fprintf('%3i %8.3f\n',a,x)`	

Siehe auch interaktives Plot Tool (Symbol  im Plot Fenster)

5. Skripte und Funktionen (siehe auch 3.1 und 3.2)

Skripte

`clear`	
`a=10`	
`whos`	
`edit mein_skript.m`	Erstelle Datei mein_skript.m im aktuellen Verzeichnis; alternativ kann die Datei über die Benutzeroberfläche erstellt werden

Fügen Sie folgende Zeilen im Editor in `mein_skript.m` ein:

```
a=3
b=4
a2=a*a
b2=b*b
c2=a2+b2
```

Speichern Sie die Datei und führen Sie im Kommondozeilenfenster folgende Befehle aus:

`mein_skript`	Aufruf des Skriptes im aktuellen Verzeichnis ohne .m Endung;
`a`	Seiteneffekt
`whos`	

Funktionen

`clear`	
`whos`	
`edit meine_funktion.m`	Erstelle Datei meine_funktion.m im aktuellen Verzeichnis

Fügen Sie folgende Zeilen im Editor in `meine_funktion.m` ein:

```
function c2=meine_funktion(a,b)
a2=a*a
b2=b*b
c2=a2+b2
return
```

Speichern Sie die Datei und führen Sie im Kommondozeilenfenster folgende Befehle aus:

`alpha=3`	
`beta=4`	
`t=meine_funktion(alpha,beta)`	Aufruf der Funktion ohne .m Endung mit den Argumenten `alpha` und `beta`
`whos`	

Anonyme Funktionen

`clear`	
`f = @(x) 1+x.^2`	Anonymen Funktion mit einer Eingabe als Variable `f` (function handle)
`whos`	
`f(10)`	
`f(1:5)`	
`g =@(x,y) x.^2 + y.^2`	Anonyme Funktion `g` mit zwei Eingabewerten
`g(2,-1)`	
`g(1:3,4:6)`	

6. Schleifen (siehe 2.3)

`clear`	
`whos`	
`edit meine_schleife.m`	Erstelle Datei `meine_schleife.m` im aktuellen Verzeichnis

Fügen Sie folgende Zeilen im Editor in `meine_schleife.m` ein:

```
t = 0
for i=1:10
  t = t + i
end
```

Speichern Sie die Datei und führen Sie im Kommondozeilenfenster folgende Befehle aus:

`meine_schleife`	
`whos`	
`t`	

Für eine Schleife kann ebenfalls `while` verwendet werden.

7. if-Anweisung (siehe auch 2.1 und 2.3)

`1==1`	Vergleichsoperator; Rückgabewert 1 (wahr) oder 0 (falsch)
`1~=1`	Operator nicht gleich
`1<=2`	Weitere Operatoren `<` , `>` , `>=`
`1<=2 && 3<=2`	Logisches Und
`1<=2 || 3<=2`	Logisches Oder
`clear`	
`whos`	
`edit mein_if.m`	Erstelle Datei `mein_if.m` im aktuellen Verzeichnis

Fügen Sie folgende Zeilen im Editor in `mein_if.m` ein:

```
function mein_if(a)
if a>10
      disp('Argument ist größer als 10');
elseif a>=0
      disp('Argument ist größer oder gleich 0 aber kleiner gleich 10');
else
      disp('Argument ist kleiner als 0‘);
end
```

Speichern Sie die Datei und führen Sie im Kommondozeilenfenster folgende Befehle aus:

`mein_if(8)`	
`mein_if(12)`	
`mein_if(-2)`	

8. Symbolische Berechnungen

`clear`	
`syms x`	Erzeugt die symbolische Variable `x`
`fs=x^2-2`	Definition der Funktion $f(x) = x^2 - 2$ mit symbolischer Variable x
`dfs = diff(fs)`	Ableitung f' der Funktion f
`fs(2)`	Fehler: Keine direkte Auswertung von Funktionen mit symbolischen Variablen
`subs(fs,x,2)`	Auswertung
`f=matlabFunction(fs)`	Konvertierung in anonyme Funktion (analog für `dfs`)
`f(2)`	Auswertung von f