Institut für Informationsverarbeitung
Leibniz Universität Hannover
http://www.tnt.uni-hannover.de

MATLAB Einführung

Inhaltsverzeichnis

1 Einführung in MATLAB

MATLAB ist eine kostenpflichtige Software mit vielen nützlichen Toolboxen. Alternativ zum vollen Kauf oder einer Studentenlizenz gibt es vergleichbare OpenSource Software, wie beispielsweise Octave, Scilab oder auch SciPy.
Eine Gegenüberstellung von MATLAB und Octave findet sich unter
https://en.wikibooks.org/wiki/MATLAB_Programming/Differences_between_Octave_and_MATLAB

1.1 Erste Schritte in der Matlabkonsole

Nachdem MATLAB gestartet wurde, können Berechnungen direkt in der Kommandozeile ausgeführt werden. Wie bei anderen Programmen, werden die einzelnen Befehle immer mit einem Zeilenumbruch (ENTER) bestätigt. Das Ergebnis der Operation wird anschließend von MATLAB ausgegeben. Dezimalzahlen werden in englischer Schreibweise mit einem Punkt angegeben.

So lässt sich MATLAB unter Verwendung der üblichen Operatoren ohne weitere Kenntnisse bereits als einfacher Taschenrechner verwenden:

```
>> 64*2^3
ans =
   512
>> 125/5 * (6.8-2.8)
ans =
   100
```

Ergebnisse können durch folgende Wertzuweisung in Variablen gespeichert werden, wobei kein Datentyp deklariert werden muss:

```
>> a = 1
a =
     1
```

Diese Variablen sind dann im lokalen Workspace gespeichert, so lange bis sie gelöscht werden oder MATLAB beendet wird. Nach jedem Befehlsende kann ein Semikolon (`;`) gesetzt werden. Das Semikolon verhindert die Ausgabe des Befehlsrückgabewertes:

```
>> a = 1;
```

Es können auch mehrere Befehle (durch Komma oder Semikolon getrennt) in einer Zeile angegeben werden:

```
>> a = 1, b = 2; c = 3;
a =
    1
```

Variablen können durch eine neue Zuweisung einfach überschrieben werden:

```
>> a = 42 - 39; b = 3*5;
>> c = a^2 + b;
>> c = c+1
c =
    25
```

Über Zahlen hinaus, können auch Zeichenketten in Variablen gespeichert werden, wobei bei der Verwendung von Hochkommata ein einzelnes Zeichen als ein Element eines Character-Arrays abgespeichert wird.

```
>> s='Hello World!'
s =
Hello World!
>> disp('Hello World')
Hello world!
```

Es bietet sich an die Kommandos in einer Datei zu speichern um wiederholt dieselbe Folge von Berechnungen durchführen zu können, insbesondere bei umfangreicheren Berechnungen. Der MATLAB Code wird üblicherweise in m-Files gespeichert, die über den MATLAB Editor erstellt, bearbeitet und abgespeichert werden können. Um ein neues m-File im Editor zu öffnen:

```
>> edit
```

bzw. um ein neues m-File mit dem Namen `funnyName.m` zu erstellen:

```
>> edit funnyName
>> edit funnyName.m
```

Dort kann beliebiger MATLAB Code platziert werden, der durch Eingabe des entsprechenden Dateinamen auch über die MATLAB Konsole ausgeführt werden kann. Im Folgenden werden exemplarisch einige MATLAB Befehle aufgelistet und erklärt, die etliche Beispiele enthalten um einen schnellen Einstieg zu ermöglichen.

1.2 Basics

Es sollte unbedingt vermieden werden, dass neue Variablen und m-Files denselben Namen erhalten wie ein bereits vorhandenes MATLAB build-in Schlüsselwort oder Funktion (z.B. `edit`, `for`, `image`, ...).

Beschreibung:	**Befehl (Beispiel):**
Hilfe im Befehlsfenster	`F1` `help functionName`
Dokumentation im Browser	`doc` `doc functionName`
Suchen einer Zeichenfolge in der ersten Zeile aller MATLAB Dateien im Suchpfad	`lookfor someWord`
Inhalt der MATLAB-Konsole löschen	`clc`
Schließt alle offenen Display Fenster	`close all`
Schließt das Display Fenster mit Handle `H`	`close(H)`
Die Variable `varName` löschen	`clear varName`
Alle Variablen im Workspace löschen	`clear all`
Alle Variablen im Workspace anzeigen	`whos`
Speichert alle Variablen des Workspace in der Datei `data.mat`	`save data.mat`, `save('data.mat')`
Speichert die Variablen `v1`, `v2` in `data.mat`	`save data.mat v1 v2` `save('data.mat','v1','v2')`
Lädt Variablen der Datei `data.mat` in den Workspace (überschreibt ggfs. gleichnamige Variablen)	`load data`, `load data.mat`, `load('data.mat')`
Öffnet den Editor	`edit`
Kommentarzeile (Strg + R)	`% explanation`
Laufende Berechnung abbrechen	`Strg + C`

1.3 Vektoren und Matrizen

In MATLAB gilt: *alles ist eine Matrix*. Das heißt, in MATLAB wird alles in (ggfs. multidimensionalen) Matrizen abgelegt. Matrizen sind *typisiert*, d.h. die Matrixelemente besitzen einen Typ aus folgender (nicht abschließender) Liste: `double`, `single`, `int8`, `uint8`, `int16`, etc. Einfache Zahlenwerte und Vektoren sind Spezialfälle von Matrizen. So ist ein M-dimensionaler Zeilenvektor eine $1 \times M$-Matrix und ein N-dimensionaler Spaltenvektor eine $N \times 1$-Matrix. Matrizen und Vektoren werden in eckigen Klammern angegeben. Die einzelnen Vektor-/Matrix-Elemente werden durch Leerzeichen oder Kommata (für nächste Spalte) bzw. Semikola (für nächste Zeile) getrennt:

Beschreibung:	**Befehl (Beispiel):**
Erstellung eines Zeilenvektors (1×5-Matrix)	`[1, 2, 3, 2, 1]` `[1 2 3 2 1]`
Erstellung eines Spaltenvektors (5×1-Matrix)	`[1; 2; 3; 2; 1]`
Erstellung einer 2×3-Matrix	`[1, 2, 3; 4, 5, 6]`
Zeilenvektor mit Zahlen von 1 bis 5	`1:5` `1:1:5`
Zeilenvektor mit Zahlen von 1 bis 5 in 0.5er Schritten	`1:0.5:5` `linspace(1,5,0.5)`
Wertezuweisung und explizite Wertausgabe	`A = [1, 2, 3, 4, 5];` `disp(A);`
Erzeugung einer $M \times N$-Nullmatrix	`B = zeros(M, N);`
Erzeugung einer $M \times N$-Einsmatrix	`B = ones(M, N);`
Erzeugung einer $N \times N$-Einheitsmatrix	`B = eye(N);`
Erzeugung einer 3×3 Matrix aus einem Vektor der Länge 3	`A = diag([1 2 3])`
Extrahieren der Matrixdiagonalen	`diag(A)`
Zugriff auf das n-te Element des Vektors	`a(n)`
Zugriff auf Zeile i und Spalte j der Matrix A	`A(i, j)`
Zugriff auf die erste Zeile der Matrix A	`A(1, :)`
Ersetzen des Elements $A_{2,1}$ von A auf 5	`A(2, 1) = 5;`
Ersetzen einer Teilmatrix	`A(3:4, 2:3) = eye(2);`
Umwandeln eines (multidimensionalen) Arrays A in einen Spaltenvektor b (*flattening*)	`b = A(:);` $\Rightarrow$ `[1,2;3,4]` wird `[1;2;3;4]`

Verkettung von Matrizen	`A = [1,2]; B = [3,4];`
	`C = [A, B] = [1, 2, 3, 4]`
	`C = [A; B] = [1, 2; 3, 4]`

1.4 Matrixoperatoren

Mit den bekannten Rechenoperatoren `+, -, *, /, ^` lassen sich Zahlen, Vektoren, Matrizen und mehrdimensionale Arrays verrechnen. Für elementweise Operationen werden vor die Operatoren für Multiplikation, Division und Potenz jeweils ein einfacher Punkt eingefügt: `A.*A`

Beschreibung:	**Befehl (Beispiel):**
Matrix-Matrix-Addition	`C = A+B;` `[1, 2] + [3, 4]`
Matrix-Skalar-Addition	`[1, 2] + 1`
Matrix-Matrix-Subtraktion	`C = A-B;` `[1, 2] - [3, 4]`
Matrix-Skalar-Subtraktion	`C = [1, 2] - 1`
Skalar-Matrix-Multiplikation	`2 * [1, 2; 3, 4]`
Matrix-Vektor-Multiplikation	`A*b = [1, 2; 3, 4]*[5; 6]`
Matrix-Multiplikation	`A*B = [1, 2; 3, 4]*[5, 6; 7, 8]`
Elementweise Multiplikation	`A.*B=[1, 2; 3, 4].*[5, 6; 7, 8]`
Elementweise Division Links: Elementweise Division Rechts:	`[1, 2; 3, 4] .\ [5, 6; 7, 8]` `[1, 2; 3, 4] ./ [5, 6; 7, 8]`
Matrix-Division: Lösung von $A \cdot x = b$ (siehe auch `help mldivide`)	`x = A\b`
Potenzieren (A^n)	`[1, 2; 3, 4]^2`
Elementweises Potenzieren	`[1, 2; 3, 4].^2`
Matrix-Transposition A^T	`[1, 2; 3, 4].'`
komplex konjugierte Transposition $\overline{A}^\mathsf{T}$	`[1, 2; 3, 4]'`
Matrix-Inversion A^{-1}	`inv([1, 2; 3, 4])`
Determinante $\lvert A \rvert$	`det([1, 2; 3, 4])`

1.5 Nützliche Matrixfunktionen

Funktionen, die quantitative Aussagen über die Ausmaße oder den Inhalt einer Datenstruktur liefern.

Beschreibung:	**Befehl (Beispiel):**
Gesamtanzahl der Elemente eines Arrays	`numel(A)` `numel([1 2])` ⇒ 2 `numel([1 2; 4 5])` ⇒ 4
Anzahl der Elemente pro Dimension	`size(A)` `size([1 2 1])` ⇒ `[1, 3]`
Länge der längsten Dimension eines Arrays	`length(A)` ⇔ `max(size(A))`
Anzahl der Dimensionen eines Arrays	`ndims(A)` ⇔ `length(size(A))`
Summe aller Vektorelemente	`sum(v)`

Weitere häufig hilfreiche Funktionen:
`min, max, mean, median, prod, round, ceil, floor`

1.6 Vergleichsoperatoren und Logik

Einfache Zahlenwerte, Vektoren, Matrizen und Arrays höherer Ordnung lassen sich mit den Vergleichsoperatoren `>, <, <=, =>, ==, ~=` vergleichen. Die Vergleiche liefern logische Werte, d.h. 0 für *false* und 1 für *true*, wobei sie angewandt auf ein

Array die entsprechenden Werte in korrespondierender Dimension zurückliefern. Zum Verarbeiten von mehreren logischen Werte können logische Operatoren wie `and` (`&`), `or` (`|`) oder `not` (`~`) verwendet werden.

Beschreibung:	**Befehl (Beispiel):**
Testet ob die Eingabe leer ist	`isempty(A)`
Testet ob eine Variable existiert	`exist varName`
Testet ob die Eingabe eine Zahl ist	`isnumeric`
Testet ob die Eingabe ein String ist	`ischar, isstr`
Liefert die Indizes der Matrixelemente von A, die kleiner sind als 25	`[row, col] = find(A < 25)`

1.7 Matrizen und Logik

Einige logische Befehle können direkt auf Matrizen angewendet werden:

Beschreibung:	**Befehl (Beispiel):**

Setzt alle Stellen, an denen ein bestimmter Wert steht, auf 1 und alle anderen auf 0	`M = [7 5; 8 5]`
	`N = (M==5)`
	`ans =`
	`0 1`
	`0 1`

Löscht doppelte Werte aus einer Matrix und sortiert die Werte	`M = [7 5 8 5]`
	`unique(M)`
	`ans =`
	`5 7 8`

1.8 Schleifen und Bedingungen

Beschreibung:	**Befehl (Beispiel):**

for-Schleife	`n = 10;`
	`a = zeros(n, 1);`
	`for id = 1:n`
	`  a(id) = id+1;`
	`end`

while-Schleife	`n = 10;`
	`a = zeros(n, 1);`
	`i = 1;`
	`while i <= n`
	`  a(i) = i+1;`
	`  i = i+1;`
	`end`

if-Anweisung	`if nargin==0`
	`  disp('No input')`
	`elseif(nargin==1)`
	`  disp('I found one input')`
	`else`
	`  disp('I found some more input')`
	`end`

switch-Anweisung	`switch day`
	`  case 'friday'`
	`  disp('so close...')`
	`  case {'saturday','sunday'}`
	`  disp('it''s weekend!')`
	`  otherwise`
	`  disp('just work...')`
	`end`

Fehler abfangen mit `try`	`try`
	`  I = imread('image.jpg');`
	`catch`
	`  disp('The image could not be read')`
	`  rethrow(lasterror)`
	`end`

1.9 Funktionen

MATLAB Funktionen werden in einem gleichnamigem m-File gespeichert und können dann mit ihrem jeweiligen Funktions- bzw. Dateinamen aufgerufen werden. Die Variablen innerhalb der Funktion werden nicht im Workspace gespeichert, d.h. sie sind nur lokal zugänglich und können über die Funktion hinaus nicht verwendet werden. Eine Funktion muss nicht unbedingt Eingabe- oder Ausgabewerte haben. Es kann nützlich sein Unterfunktionen in einer vorhandenen Funktion zu definieren, um beispielsweise Code-Wiederholungen zu reduzieren.
Achtung: Es ist zu vermeiden, dass neue Funktionen denselben Namen erhalten wie bereits vorhandenen MATLAB build-in Funktionen (gilt auch für Variablen).

Beschreibung:	**Befehl (Beispiel):**

Funktion mit dem Namen `combineValues`	`function[out1, out2] = ...`
	`  combineValues(in1, in2)`
	`out1 = in1 + in2;`
	`out2 = in1 - in2;`
	`end`

Anzahl der Eingabeparameter	`nargin`
Anzahl der Ausgabeparameter	`nargout`
Funktionsabbruch mit Fehlermeldung `'message'`	`error('message')`
Die globale Variable steht allen Funktionen zur Verfügung, welche sie deklarieren	`global varName`
lokale Variable, deren Wert auch zwischen den Funktionsaufrufen im Speicher abgelegt wird	`persistent varName`

1.10 Nützliche Datenstrukturen

Um das Abspeichern verschiedener Datentypen in einer Variablen zu ermöglichen bedarf es spezifischer Datenstrukturen. Ein Cell-Array ist die Erweiterung eines einfachen Arrays, deren Elemente weiterhin über einen Index abgerufen werden können. In einem Struct-Array werden die Informationen durch spezifische Feldnamen abgespeichert.

Beschreibung:	**Befehl (Beispiel):**
Erstellung eines leeren Cell Arrays	`a = cell(1, 3);  % empty cell`

Cell Array mit drei Elementen	`person = {'Herbert', 42, [ 1 3 9 ]};`
	`>> person{2}`
	`ans = 42`

Struct Array	`personstruct = struct('name', ...`
	`  'Tobi', 'age', 21);`
	`>> personstruct.age`
	`ans = 21`

1.11 Umwandlung von nützlichen Datenstrukturen

Um Daten weiter zu verarbeiten, ist es oft auch notwendig, verschiedene Datenstrukturen in einander umzuwandeln.

Beschreibung:	**Befehl (Beispiel):**

Ein Struct in ein Cell Array umwandeln	`struct2cell(personstruct)`
	`ans =`
	`  {'Tobi'}`
	`  {[  21]}`

Ein Cell Array in eine Matrix umwandeln. Dafür müssen in den Cells gleiche große Matrizen stehen	`A = {[5 4; 3 2], [6 7; 8 9]}`
	`cell2mat(A)`
	`ans =`
	`  5  4  6  7`
	`  3  2  8  9`

Aus Zahlen bzw. Matrizen wird eine neue Matrix gebildet. Dabei werden die Werte Spaltenweise entnommen und zeilenweise geschrieben.	`reshape(A,4,2)`
	`ans =`
	`  5  6`
	`  3  8`
	`  4  7`
	`  2  9`

1.12 Grafik

Folgende Funktionen sind nützlich, um mit Bildern und grafischen Darstellungen in MATLAB zu arbeiten.

Beschreibung:	**Befehl (Beispiel):**
Öffnet eine neue Figure (Fenster) und gibt das Handle (die Referenz) zurück	`H = figure;`
Öffnet das Fenster mit Handle/ID `H`	`figure(H)`
Handle des aktuellen Fensters	`gcf`
Löscht den Inhalt des aktuellen Fensters	`clf`
Schließt alle offenen Fenster	`close all`
Titel für die aktuelle Figure	`title('image name')`
Achsenbeschriftungen	`xlabel('x'), ylabel('y')`
Inhalt der aktuellen Figure wird [nicht] ersetzt	`hold on` [`hold off`]
Unterteilung des Grafikfensters	`subplot(rows, cols, id)`
Bild einlesen	`I = imread('image.jpg');`
Bild darstellen	`imshow(I)`
Plottet Wertpaare von (x, y) in ein kartesisches Koordinatensystem im aktuellen Fenster	`plot(1, 3)` `plot(1:10, sin(1:10))`

1.13 Editor Einstellungen und Shortcuts

Shortcuts (Tastenkombinationen) sind ein nützliches Hilfsmittel für alle Programmierer. Da der Editor beim Starten von MATLAB oft nicht so eingestellt ist, dass intuitive Shortcuts zur Verfügung stehen (insb. unter Linux, außer man ist an Emacs gewohnt), sollte dies vor der Arbeit im MATLAB Editor überprüft werden: Die Einstellungen lassen sich im Tab "Home" über die Schaltfläche "Preferences" (kleines Zahnradsymbol) öffnen. Unter *MATLAB* → *Keyboard* → *Shortcuts* → *Active Settings* befindet sich die Einstellung für die Default Shortcuts.

Es folgt eine kurze Übersicht mit einigen verfügbaren Tastenkombinationen des MATLAB Editors, die vollständige Liste ist in den Einstellungen zu finden.

Tastenkombination	**Wirkung**
Strg + S	Speichern des aktuellen Editor Inhalts in einer Datei ggfs. wird eine dafür eine neue Datei erzeugt
Strg + C	Kopieren des markierten Texts
Strg + X	Entfernen und Kopieren des markierten Texts
Strg + V	Einfügen des zuletzt markierten Texts
Strg + R	Die markierten Zeilen werden kommentiert: `%` wird am Anfang der Zeile hinzugefügt `%%` leitet einen neuen Abschnitt ein
Strg + T	Die markierten Zeilen werden auskommentiert: Falls `%` am Anfang der Zeile steht, wird es entfernt
Strg + I	Automatische Code-Einrückung (Smart-Indent)

Strg + Enter	In einem Skript (nicht innerhalb einer Funktion!) wird der aktuelle Abschnitt ausgeführt
F1	Hilfe: Öffnen der Dokumentation
F5	Auswertung des Codes im der Datei
F9	Auswerten des Codes, der markiert ist
F12	Setze einen Breakpoint in die aktuelle Zeile

Weitere möglicherweise hilfreiche Einstellungen befinden sich unter

- *MATLAB* → *Command Window* → *Text display* → *Numeric format*
  ⇒ Darstellung numerischer Werte im *Command Window* anpassen (geht z.B. auch über das Kommando `format shortG`)
- *MATLAB* → *Variables* → *Format*
  ⇒ Darstellung numerischer Werte im Workspace anpassen
- *MATLAB* → *Editor/Debugger* → *Language* → *Comment formatting*
  ⇒ Wrap comments automatically
- *MATLAB* → *Editor/Debugger* → *Code Folding*
  ⇒ für If/else blocks

Der MATLAB Editor verfügt zudem leider über keine eigenen Code Formatter. Diese Funktion kann jedoch über Skripte wie MBeautifier nachgerüstet werden.