NTB
INTERSTAATLICHE HOCHSCHULE
FÜR TECHNIK BUCHS

Weiterbildungs-Kurs

MATLAB – Grundlagen

Referenz

Michael Schreiner

Inhaltsverzeichnis

1 Grundlagen: Variablen und Workspace

Variablennamen

- case-sensitive
- Bis zu 31 Zeichen lang
- Beginnen mit einem Buchstaben
- Buchstaben, Ziffern, Underscore

Spezielle Variablen

`ans`	Default–Variable für Resultate
`beep`	Beep
`pi`	$\pi = 3.14\ldots$
`eps`	Die kleinste Zahl, so dass $1 + \text{eps} \neq 1$
`inf`	Unendlich (z.B. $1/0$)
`NaN` oder `nan`	Not-a-Number (z.B. $0/0$)
`i,j`	Imaginäre Einheit ($\sqrt{-1}$)
`nargin`	Anzahl Input-Argumente bei Funktionen
`nargout`	Anzahl Output-Argumente bei Funktionen
`realmin`	Kleinste positive reelle Zahl
`realmax`	Größte reelle Zahl
`bitmax`	Größte ganze Zahl
`varargin`	Optionale Input–Argumente
`varargout`	Optionale Output–Argumente

Kommentare und Co.

`% Kommentar`	Kommentare sehen so aus
`;`	Abschluss eines Befehls. Ausgabe wird unterdrückt
`,`	Trennt Befehle
`...`	Befehl wird in nächster Zeile fortgesetzt
`Ctrl+C`	Matlab-Prozess wird unterbrochen

Zahlenformate

Befehl	π	Beschreibung
`format short`	`3.1416`	Default–Format
`format short e`	`3.1416e+000`	Exponential–Darstellung
`format short g`	`3.1416`	Das Beste der beiden
`format long`	`3.14159265358979`	Genaue Darstellung
`format long e`	`3.141592653589793e+000`	... in Exponentialform
`format long g`	`3.14159265358979`	Das Beste der beiden
`format bank`	`3.14`	2 Nachkommastellen
`format hex`	`400921fb54442d18`	Hexadezimale Floating–Point–Zahl
`format rat`	`355/113`	Rationale Approximation

Workspace–Kommandos

`who`	Welche Variablen sind vorhanden?
`whos`	Variablen mit Speicherplatzangabe
`clear a`	Löschen der Variable a
`clear all`	Alle Variablen im Workspace werden gelöscht
`help`	Hilfe–Übersicht (Kommandozeile)
`help Befehl`	Hilfe zu Befehl (Kommandozeile)

2 Mathematische Operatoren und Funktionen

Mathematische Operatoren

Befehl	Beispiel	Beschreibung
+	47 + 11	Addition
-	3 - pi	Subtraktion
*	2*4	Multiplikation
/ oder \	21.3/7 oder 7\21.3	Division
^	2^3	Potenzieren

Vergleiche, Logische Operatoren

x < y	kleiner als
x <= y	kleiner gleich
x > y	größer als
x >= y	größer gleich
x == y	ist gleich
x ~= y	ist ungleich
x & y	logisches Und
x | y	logisches Oder
~x	nicht

Trigonometrische Funktionen und Co.

```
sin(x),   cos(x),   tan(x),   cot(x)
asin(x),  acos(x),  atan(x),  acot(x)
atan2(y,x)                                  −π < atan2(y,x) ≤ π
sinh(x),  cosh(x),  tanh(x),  coth(x)
asinh(x), acosh(x), atanh(x), acoth(x)
```

Exponentialfunktion und Co.

`x^y`	x^y
`exp(x)`	Exponentialfunktion (Basis e)
`log(x)`	natürlicher Logarithmus
`log10(x)`	Logarithmus zur Basis 10
`log2(x)`	Logarithmus zur Basis 2
`sqrt(x)`	Wurzelfunktion

Funktionen für komplexe Rechnungen

`abs(x)`	$\|x\|$
`angle(x)`	$\arg x$
`conj(x)`	konjugiert komplexe Zahl
`imag(x)`	Imaginärteil
`real(x)`	Realteil
`complex(x,y)`	$x + iy$

Runden etc.

`fix(x)`	Runden zur Null
`floor(x)`	Runden zu $-\infty$
`ceil(x)`	Runden zu $+\infty$
`round(x)`	Runden zur nächsten ganzen Zahl
`mod(x)`	Vorzeichenbehafteter Divisionsrest
`rem(x)`	Positiver Divisionsrest
`sign(x)`	Signum–Funktion

Andere nützliche Funtionen

`[th,phi,r] = cart2sph(x,y,z)`	Transformation kartesische in Kugelkoordinaten (`th`: Länge, `phi`: Breite, `r`: Radius)
`[x,y,z] = sph2cart(th,phi,r)`	Umkehr–Transformation
`[th,r,z] = cart2pol(x,y,z)`	Transformation kartesische in Zylinderkoordinaten
`[th,r] = cart2pol(x,y)`	Transformation kartesische in Polarkoordinaten
`[x,y,z] = pol2cart(th,phi,r)` `[x,y] = pol2cart(th,r)`	Umkehr–Transformation
`factor(x)`	Primfaktorzerlegung

3 Arrays

3.1 Array–Konstruktionen

Array–Konstruktion (Zeilenvektoren)

`x = [2 pi sqrt(2)]`	Zeilenvektor mit definierten Elementen
`x = [2, pi, sqrt(2)]`	Zeilenvektor mit definierten Elementen
`x = first:last`	Erzeugt einen Zeilenvektor. Start mit `first`, Ende bei `last` oder davor. Abstand der Elemente ist 1.
`x = first:increment:last`	Wie zuvor, aber Abstand der Elemente ist `increment`
`x= linspace(first, last, n)`	Zeilenvektor mit n Elementen von `first` bis `last`.
`x= linspace(first, last)`	Wie zuvor mit 100 Elementen
`x= logspace(first, last, n)`	Zeilenvektor mit n Elementen von $10^{\texttt{first}}$ bis $10^{\texttt{last}}$.

Array–Konstruktion (Spaltenvektoren)

`x = [2;pi;sqrt(2)]`	Spaltenvektor mit definierten Elementen
`x = [2, pi, sqrt(2)]'`	Spaltenvektor durch Transponierung eines Zeilenvektors
`x = (first:last)'`	Spaltenvektor durch Transponierung

Hinweise zum Transponier–Operator

- Der Operator ' ist eigentlich ein Adjungier–Operator, das heißt er transponiert und konjugiert komplexe Zahlen.
- Will man einen Vektor aus komplexen Zahlen nur transponieren, so ist der Operator `.'` zu verwenden.
- Für reelle Vektoren ist ' und `.'` identisch. In der Praxis wird daher meist ' verwendet.

Array–Konstruktion (Matrizen)

- Matrizen sind zweidimensionale Arrays.
- In Matlab werden Vektoren immer als Matrizen angesehen. Ein Zeilenvektor der Länge n ist eine $1 \times n$–Matrix, ein Spaltenvektor mit m Elementen eine $m \times 1$–Matrix.
- Matrizen erzeugt man durch Zusammensetzen von Vektoren oder anderen Matrizen.
- Eingabe: Trennung von Spalten mit Komma oder Space, Trennung von Zeilen mit Strichpunkt oder Return.

Array–Konstruktion (Spezielle Matrizen)

`A = zeros(r,c)`	$r \times c$–Array mit Nullen
`A = ones(r,c)`	$r \times c$–Array mit Einsen
`A = eye(n)`	$n \times n$–Array mit Einheitsmatrix
`A = eye(r,c)`	$r \times c$–Array mit Einheitsmatrix
`A = rand(r,c)`	$r \times c$–Array mit gleichverteilten Zufallszahlen in $[0, 1]$
`A = rand(n)`	$n \times n$–Array mit gleichverteilten Zufallszahlen in $[0, 1]$
`A = randn(r,c)`	$r \times c$–Array mit standardnormalverteilten Zufallszahlen
`A = randn(n)`	$n \times n$–Array mit standardnormalverteilten Zufallszahlen

3.2 Array–Zugriff

- Indizierung beginnt bei 1 (nicht bei 0)
- Elementzugriff bei Vektoren: `x(i)`. Für Zeilen– und Spaltenvektoren
- Elementzugriff bei Matrizen: `A(r,c)`. Element in Zeile r und Spalte c
- Da jeder Vektor auch als Matrix angesehen wird, geht auch `x(i,1)` bei Zeilenvektoren und `x(1,i)` bei Spaltenvektoren.
- Eine ganze Zeile oder Spalte einer Matrix bekommt man mit `A(r,:)` (Zeile r als Zeilenvektor) oder `A(:,c)` (Spalte c als Spaltenvektor).
- Letztes Element hat Index `end`.
- Zugriff geht auch mit Indexvektoren.
- Bei Matrizen kann auch mit nur einem Index zugegriffen werden (`A(3)` oder `A(:)`). In diesem Fall werden alle Elemente der Matrix Spaltenweise zu einem großen Spaltenvektor zusammengefasst (absolute Adressierung).

3.3 Mathematik mit Arrays

Operationen und Funktionen mit Arrays

Variablen: $A = [a_1, \ldots a_n]$, $B = [b_1, \ldots, b_n]$, c Skalar

`sin(A)`	$[\sin a_1, \ldots, \sin a_n]$
`A+c`	$[a_1 + c, \ldots, a_n + c]$
`A-c`	$[a_1 - c, \ldots, a_n - c]$
`A*c`	$[a_1 c, \ldots, a_n c]$
`A/c`	$[a_1/c, \ldots, a_n/c]$
`A+B`	$[a_1 + b_1, \ldots, a_n + b_n]$
`A.*B`	$[a_1 b_1, \ldots, a_n b_n]$
`A./B`	$[a_1/b_1, \ldots, a_n/b_n]$
`B.\A`	$[a_1/b_1, \ldots, a_n/b_n]$
`A.^c`	$[a_1^c, \ldots, a_n^c]$
`c.^A`	$[c^{a_1}, \ldots, c^{a_n}]$
`A.^B`	$[a_1^{b_1}, \ldots, a_n^{b_n}]$

3.4 Array–Manipulationen und Array–Größe

Array–Manipulationen

`A = diag(x)`	Diagonalmatrix mit Diagonalelementen aus dem Vektor x
`x = diag(A)`	Vektor der Diagonalelemente von A
`diag(diag(A))`	Alle Nicht–Diagonalelemente von A werden zu null gesetzt
`triu(A)`	Obere Dreiecksmatrix von A
`tril(A)`	Untere Dreiecksmatrix von A
`flipud(A)`	Matrix wird horizontal gespiegelt
`fliplr(A)`	Matrix wird vertikal gespiegelt
`rot90(A)`	Matrix wird um 90° gegen den Uhrzeigersinn gedreht
`reshape(A,r,c)`	Matrix wird auf neue Größe $r \times c$ gebracht. Anzahl der Elemente muss kompatibel sein, d.h. A muss $r \cdot c$ Elemente besitzen.
`repmat(A,[m n])`	Die Matrix A wird $m \times n$ mal repliziert.
`repmat(A,m,n)`	Wie `repmat(A,[m n])`

Array–Größe

`s = size(A)`	Vektor mit Anzahl Zeilen und Anzahl Spalten
`[r,c] = size(A)`	r: Anzahl der Zeilen, c: Anzahl der Spalten
`r = size(A,1)`	Anzahl der Zeilen
`c = size(A,2)`	Anzahl der Spalten
`m=max(size(A))`	Die Größere der Zeilen– und Spaltenanzahl
`n=length(A)`	Wenn A nicht leer ist, ist $n =$`max(size(A))`. Wenn A null Zeilen oder null Spalten hat, ist $n = 0$.
`n=numel(A)`	Anzahl der Elemente von A.

3.5 Sortieren und Finden

Sortieren

`xs = sort(x)`	Sortiert den Vektor x aufsteigend
`[xs, idx] = sort(x)`	Sortiert und liefert den Index–Vektor `idx` zurück, so dass `x(idx)` der sortierte Vektor ist.
`As = sort(A)`	Sortiert die Matrix A spaltenweise
`[As, idA ] = sort(A)`	Wie oben. Zusätzlich enthält die Matrix `idx` spaltenweise die Zeilenindizes.
`[As, idA ] = sort(A,1)`	Sortiert spaltenweise, wie `[As, idA ]= sort(A)`
`[As, idA ] = sort(A,2)`	Sortiert zeilenweise.

Finden

`idx = find(A)`	Liefert die Indizes i, für die $A_i \neq 0$ ist (d.h. „wahr“ in Matlab–Terminologie). Ist A eine Matrix, wird A als Spaltenvektor (absolute Adressierung) interpretiert.
`[r,c] = find(A)`	Liefert die Vektoren r und c mit den Zeilen und Spaltenindizes der Elemente von der Matrix A, die ungleich null sind.

3.6 Implizite Array–Erzeugung

Wenn ein $r \times c$–Array erzeugt wurde, und es wird versucht, auf ein Element außerhalb des Arrays zuzugriefen, sind zwei Fälle zu unterscheiden:

- Wenn **lesend** zugegriffen wird (`c=A(n,m)`), wird der Versuch mit einer Fehlermeldung honoriert.
- Wird **schreibend** zugegriffen (`A(n,m) = 1`), dann wird das Array entprechend vergrößert, und die neuen Elemente werden mit null initialisiert.

4 Skript–m–Files

4.1 Einführung

- Matlab-Befehle können in einem File mit der Endung `.m` (m–File) abgespeichert werden.
- Aufruf im Kommando–Window durch Eingabe des Filenamens (ohne `.m`)
- Kommandos funktionieren wie im Workspace.
- Zugriff auf die gleichen Variablen wie im Workspace (keine lokale Variablen).
- Es gibt in Matlab einen Suchpfad. Dort sind die Directories angegeben, wo nach m–Files gesucht wird.
- Im Suchpfad ist auch die Suchreihenfolge festgelegt.

4.2 Ausführung von m–Files

Reihenfolge der Auflösung eines Namens

- Variablen–Name im Workspace
- Eingebaute Funktion
- m-File im „current directory“
- m–File im Suchpfad

Files, Directories und Pfad

`pwd`	Print Working Directory. Angabe des aktuellen Verzeichnisses.
`s = pwd`	Aktuelles Verzeichnis wird in `s` gespeichert.
`cd name`	Wechsle das aktuelle Verzeichnis zu `name` (relativ zum aktuellen Pfad).
`dir`	Files im aktuellen Verzeichnis.
`dir *m.`	Listing der m–Files im aktuellen Verzeichnis.
`path`	Zeigt den aktuellen Suchpfad.
`p = path`	Speichert den aktuellen Suchpfad in `p`.
`path(p1,p2)`	Hängt die Pfade `p1` und `p2` zusammen. Das Ergebnis ist der neue Suchpfad.
`path(path,p)`	Ergänzt den aktuellen Suchpfad um `p`
`addpath dir1 dir2 ...`	Ergänzt den Suchpfad durch die Verzeichnisse `dir1 ...`. Ergänzung am Anfang.
`addpath dir1 dir2 ... -BEGIN`	Wie oben.
`addpath dir1 dir2 ... -END`	Wie oben. Ergänzung am Ende des Suchpfades.
`addpath(pwd)`	Suchpfad wird durch aktuelles Verzeichnis ergänzt.
`rmpath dir1 dir2 ...`	Entfernen der Verzeichnisse `dir1 dir2 ...` aus dem Suchpfad.

4.3 Nützliche Kommandos in m–Files

Nützliche Kommandos in m–Files

`%`	Die ersten Kommentarzeilen von `mFile` werden ausgegeben, wenn der Benutzer `help mFile` eingibt.
`beep`	Beep.
`disp(a)`	Das Array wird ohne Array–Namen ausgegeben.
`pause(n)`	Auführung wird für n Sekunden unterbrochen.
`pause`	Ausführung wird unterbrochen bis ein Tastendruck erfolgt.
`waitforbuttonpress`	Ausführung wird unterbrochen bis ein Tastendruck oder ein Mausklick in einer Figure erfolgt.
`r = input('Text')`	Schreibt „Text“ und wartet auf die Eingabe von r. Die Eingabe wird zunächst im Workspace ausgewertet, so dass auch Operationen verwendet werden können.
`s = input('Text','s')`	Wie zuvor, erwartet aber einen String als Eingabe. Blanks werden als Blanks eingefügt.

4.4 Debugging

Kommandos für des Debuggen

`keyboard`	Das m–File wird unterbrochen. Nun kann im Kommando–Fenster gearbeitet werden. Das m–File wird weiter ausgeführt, nachdem man `return` (das heißt 'r', 'e', 't', 'u', 'r', 'n', 'Enter') eingegeben hat.
`pause off`	pause–Kommandos werden ignoriert.
`pause on`	pause–Kommandos werden beachtet.
`echo on`	Alle Kommandos in m–Files werden ausgegeben
`echo off`	Das ist der Normalzustand
`echo`	echo an/aus im Toggle–Mode
`echo file on`	Wie oben für das File `file`
`echo file off`	
`echo file`	
`which Kommando`	Gibt an, wie Kommando interpretiert wird, und wo das ausgeführte File steht.

Tipps für das Debuggen

- Breakpoints setzen
- Mit `echo on` die Ausführung beobachten.
- Überprüfen, ob Änderungen im File bereits gespeichert sind.
- Mit `which` überprüfen, welches File ausgeführt wird.

5 Zweidimensionale Plots

5.1 Das Kommando Plot

- Das Standard–Kommando zum Plotten ist `plot`.
- Plotten ist Vektor–orientiert.
- Funktionen werden geplottet, indem zuerst ein Vektor mit Funktionswerten (und evt. ein Vektor mit Abszissenwerten erzeugt wird.

Das Kommando plot

`plot(y)`	Plot der Ordinatenwerte `y`.
`plot(x,y)`	Plot der Ordinatenwerte `y` über den Abszissen `x`.
`plot(x1,y1,x2,y2,...)`	Mehrere Kurven in ein Bild.
`plot(A)`	Plot mehrerer Kurven aus den Spalten von `A`.
`plot(x,A)`	Plot mehrerer Kurven aus `A`. Je nach dem Format von `x` wird `A` spalten– oder zeilenweise interpretiert.

5.2 Farben, Linien, Symbole

- Der Plot–Befehl kann nach den Daten–Vektoren durch einen String ergänzt werden, der eine beliebige Kombination aus Farb–, Linien– und Symbolinformationen enthält.
- Beispiele:
  - `plot(y,'b-*')`
  - `plot(x,y,'g:v')`
  - `plot(x,y,'y-',x1,y1,'s')`

Farben und Linien

`b`	`[0,0,1]`	blau	`-`	durchgezogen
`g`	`[0,1,0]`	grün	`:`	gepunktet
`r`	`[1,0,0]`	rot	`-.`	strich–punkt
`c`	`[0,1,1]`	cyan	`--`	gestrichelt
`m`	`[1,0,1]`	magenta		
`y`	`[1,1,0]`	gelb		
`k`	`[0,0,0]`	schwarz		
`w`	`[1,1,1]`	weiß		

Symbole

`.`	Punkt	`v`	Dreieck
`o`	Kreis	`^`	Dreieck
`x`	Kreuz	`<`	Dreieck
`+`	Plus	`>`	Dreieck
`*`	Stern	`p`	Pentagramm
`s`	Quadrat	`h`	Hexagramm
`d`	Raute (diamond)		

Plot–Optionen

- Eine weitergehende Beeinflussung der Darstellung ist mit Plot–Optionen möglich.
- Beispiel: `plot(x,y,'r-', 'Property1', Value1, 'Property2', Value2,...)`

Weitere Plot–Optionen

`'LineWidth'`	Skalarer Wert, der die Liniendicke angibt.
`'Color'`	Farbe der Linein im RGB–Format (`[r,g,b]`).
`'MarkerSize'`	Skalarer Wert, der die Größe der Marker angibt.
`'MarkerFaceColor'`	Farbdefinition oder `'none'` (Marker–Fläche).
`'MarkerEdgeColor'`	Farbdefinition oder `'none'` (Marker–Kanten).

5.3 Titel, Beschriftungen, etc.

Titel, Beschriftungen, etc.

`title('Text')`	Einzeiliger Titel der Grafik.
`title({'Zeile 1','Zeile 2'})`	Mehrzeiliger Titel.
`box on`	Umrandung wird eingeschaltet.
`box off`	Umrandung wird ausgeschaltet.
`box`	Umrandung im Toggle–Mode.
`grid on`	Gitter einschalten.
`grid off`	
`grid`	
`text(x,y,'Text')`	Text an Position (x, y).
`xlabel('Text')`	Beschriftung der x–Achse.
`ylabel('Text')`	Beschriftung der y–Achse .
`legend('Text1','Text2',..)`.	Legende.

5.4 Axis — Anpassen der Achsen

Axis — Anpassen der Achsen

`axis([xmin xmax ymin ymax])`	Plot–Bereich festlegen.
`V = axis`	Liefert aktuellen Plot-Bereich.
`axis tight`	Plot–Bereich durch Grenzen der Daten
`axis auto`	Plot–Bereich der Achsen automatisch (default)
`axis manual`	Einstellung der Achsen werden auch bei nachfolgenden Plots nicht geändert (`hold on`).
`axis equal`	Gleicher Maßstab in x– und y–Richtung
`axis square`	Quadratische Achsen
`axis ij`	Matrix–Mode, d.h. y–Werte von oben nach unten.
`axis xy`	Matrix–Mode aufheben, also y–Werte von unten nach oben.
`axis normal`	Bild im maximalen Bereich ohne Restriktionen.
`axis on`	Achsen zeichnen.
`axis off`	Keine Achsen zeichnen.

- `axis`–Befehle wirken auf aktuellen Plot, daher werden die `axis`–Befehle nach `plot` verwendet.
- Mehrere Parameter können in einem Befehl übergeben werden, z.B. `axis on xy equal`

5.5 Mehrfachplots, etc.

Mehrfachplots, etc.

`hold on`	Alle weiteren Plot–Befehle in den aktuellen Achsen, ohne die vorhandenen Grafiken zu löschen.
`hold off`	Der nächste plot-Befehl löscht zunächst die alte Grafik (default).
`hold`	`hold on/off` im Toggle-Mode.
`subplot(n,m,p)`	Erzeugt ein $n \times m$–Array von Achsen. p gibt die Achsen an, wo der nächste Plot stattfindet. Numerierung ist zeilenweise.
`subplot(1,1,1)`	Herstellen des Default–Zustandes.
`h = figure`	Erzeugt neues Fenster. Rückgabewert h ist die Nummer (Handle).
`figure(n)`	Macht Fenster n zum aktuellen Fenster.
`h = gcf`	Liefert die Nummer, des aktuellen Fensters (get current figure).
`clf`	Löscht das aktuelle Fenster (clear figure)
`close`	Schließt das aktuelle Fenster.
`close n`	Schließt das Fenster n.
`close all`	Schließt alle Fenster.
`set(h,'Name','Text')`	Definiert den Titel des Grafik–Fensters h.

5.6 Spezielle Plot–Befehle

Spezielle Plot–Befehle

`area(x,y)`	Wie `plot`, wobei der Bereich zwischen der Kurve und der x–Achse ausgefüllt wird.
`fill(x,y)`	Ausgefülltes Polygon, das durch die Eckpunkte in x und y definiert ist.
`pie(x)`	Tortendiagramm mit Daten in a
`pie(x,e)`	Tortendiagramm. e gibt an, welche Stücke herausgestellt werden.
`plotyy(x1,y1,x2,y2)`	Plot von Daten mit unterschiedlichen y–Achsen.
`bar(x,y)`	Säulendiagramm.
`barh(x,y)`	Säulendiagramm (waagrecht).
`stairs(x,y)`	Treppenstufen–Diagramm.
`errorbar(x,y,e)`	Plot mit Fehler–Indikator.
`scatter(x,y,area)`	Scatter–Plot.
`polar(t,r)`	Polarplot, Winkel: t, Radius: r.

6 Dreidimensionale Plots

6.1 Linien–Plots

Das Kommando plot3

`plot3(x,y,z)`	Plot der Linie, die durch die Punkte in x, y und z definiert ist.
`plot3(x,y,z,s)`	`s` gibt wie beim Befehl `plot` Eigenschaften der Linien und Symbole an.

6.2 3D–Darstellung von Funktionen in zwei Variablen

- Um $f(x, y)$ darstellen zu können, müssen die Daten in Arrays vorliegen.
- Die Höheninformation muss in einer Matrix vorliegen.
- Außerdem sollten die x und y–Werte ebenfalls in zweidimensionalen Arrays vorliegen.
- Zur Erzeugung der x– und y–Arrays dient das Kommando `meshgrid`.

Das Kommando meshgrid

`[X,Y] = meshgrid(x,y)`	Erzeugt aus den Koordinaten–Vektoren x und y Matrizen X und Y mit den x– und y–Werten.

3D–Plot–Kommandos

`mesh(Z)`	Gitternetzdarstellung der Werte im zweidimensionalen Array Z.
`mesh(x,y,Z)`	Wie zuvor. x und y sind Vektoren mit den x– und y–Koordinaten. Z muss so viele Zeilen haben, wie y Einträge hat. Analog ist die Anzahl der Spalten von Z gleich der Länge von x.
`mesh(X,Y,Z)`	Gitternetzdarstellung von Z über den 2d-Arrays X und Y. Diese werden in der Regel mit `meshgrid` erzeugt.
`hidden on`	Mit Entfernung unsichtbarer Linien.
`hidden off`	Ohne Überprüfung der Sichtbarkeit.
`meshc(X,Y,Z)`	Mesh zusammen mit Isolinien.
`surf(X,Y,Z)`	Farbige Flächen–Darstellung (surface plot).
`shading faceted`	Konstante Farbe für jeden Patch. Mit Linien.
`shading flat`	Konstante Farbe für jeden Patch.
`shading interp`	Glatte Farbverläufe.
`surfc(X,Y,Z)`	Surface–Plot mit Kontur–Plot.
`surfl(X,Y,Z)`	Surface Plot mit Beleuchtung (einfach).

6.3 Isolinien–Plots von Funktionen in zwei Variablen

Isolinien–Plots

`contour(X,Y,Z,n)`	Isolinien–Plot mit n Iso–Levels.
`contour3(X,Y,Z,n)`	Isolinien–Plot mit n Iso–Levels. Darstellung in 3D.
`pcolor(X,Y,Z)`	2D–Farbdarstellung (pseudocolor). Am besten mit `shading interp`.
`contourf(X,Y,Z)`	Kombination aus Isolinien– und Farb–Darstellung.

nach `C = contour...` bzw. `[C,h] = contour...`

`clabel(C)`	Beschriftung der Isolinien.
`clabel(C,'manual')`	Interaktive Beschriftung der Isolinien.
`clabel(C,h)`	Beschriftung innerhalb der Isolinien.
`clabel(C,h,'manual')`	Interaktive Beschriftung innerhalb der Isolinien.

6.4 Spezielle dreidimensionale Plot–Kommandos

Spezielle Plot–Komandos

`quiver(X,Y,pX,pY)`	Darstellung eines Vektorfeldes. Für jeden Punkt in `X` und `Y` ist ein Vektor in `pX` und `pY` definiert.
`quiver(X,Y,pX,pY,'.')`	Darstellung eines Vektorfeldes ohne Pfeilspitzen.
`fill3(x,y,z,'c')`	Erzeugt ein Polygon mit der Farbe in `'c'`, das durch die Koordinaten definiert ist. Das Polygon ist immer geschlossen, so dass der Anfangspunkt nicht dupliziert werden muss (kann aber). Statt der festen Farbe können auch Farbverläufe durch die Vorgabe der Eckfarben erzeugt werden.

7 Start und Stop von Matlab

7.1 Start

- Beim Start von Matlab werden zwei m–Files ausgeführt: `matlabrc.m` und `startup.m`.
- `matlabrc.m` sollte nicht geändert werden.
- In `startup.m`. können eigene Anpassungen vorgenommen werden.
- `startup.m`. wird von `matlabrc.m` aufgerufen und sollte im Suchpfad stehen.
- Typischer Ort für Single–User–Installationen: `toolbox\local`.
- Für Netzwerk–Installationen: Das normale Start–Verzeichnis.

Anwendungen:

- Setzen von eigenem Pfad.
- Ändern von Standardeinstellungen.

Ein schlechtes Beispiel: startup.m

```
1 % startup.m
2 % So sollte das Startup-File NICHT aussehen
3
4 exit
```

7.2 Stop

- Nach den Kommandos `exit` bzw. `quit` wird das File `finish.m` ausgefürt.
- Einen Abbruch des Programm–Endes erreicht man mit `quit cancel`.

8 Lineare Algebra

8.1 Vektor– und Matrix–Operationen

Vektor–Operationen

`dot(x,y)`	Skalarprodukt der Vektoren `x` und `y`.
`norm(x)`	Betrag des Vektors `x`.
`cross(x,y)`	Vektorprodukt der Vektoren `x` und `y`.
`x*y`	Produkt Vektor mal Vektor (Dimensionen müssen passen)
`A*x`	Produkt Matrix mal Vektor (Dimensionen müssen passen)
`A^n`	Matrix–Exponent (quadratische Matrizen)
`e = eig(A)`	Eigenwerte der quadratischen Matrix A
`[V,D] = eig(A)`	Eigenwerte und Eigenvektoren
`det(A)`	Determinante von A.

8.2 Lineare Gleichungssysteme

Lineare Gleichungssysteme

`A\b`	Verallgemeinerte Lösung des Gleichungssystems $Ax = b$, wobei x und b Spaltenvektoren sind.
`b/A`	Verallgemeinerte Lösung des Gleichungssystems $xA = b$, wobei x und b Zeilenvektoren sind.
`inv(A)`	Inverse der Matrix A
`cond(A)`	Kondition der Matrix A
rref(A)	reduzierte Zeilennormalform (reduced row echelon form)
`rank(A)`	Rang von A

9 Funktionen

9.1 Erste Beispiele

- Funktionen sind m–Files, die als „Black–Box“ arbeiten.
- Sie haben Input– und Output–Argumente
- Sie haben einen lokalen Workspace, d.h.
  - Innerhalb der Funktion hat man keinen Zugriff auf die Variablen des globalen Workspace oder anderer Funktionen
  - Auf die lokalen Variablen in einer Funktion kann nicht von außen zugegriffen werden.

9.2 Kontroll–Strukturen

Kontroll–Strukturen

`for i=1:10`
`  (commands)`
`end`
	For–Schleife: $i = 1, \ldots, 10$.

`for i=10:-1:1`
`  (commands)`
`end`
	For–Schleife: $i = 10, 9, \ldots, 1$.

`for i=[2 5 4]`
`  (commands)`
`end`
	For–Schleife: $i = 2, 5, 4$.

`for i=[1 1 2 3]`
`  (commands)`
`end`
	For–Schleife: $i = 1, 1, 2, 3$.

`for x=[pi 4.5 7/6]`
`  (commands)`
`end`
	For–Schleife: $x = \pi, 4.5, 1.1667$.

`while expression`
`  (commands)`
`end`
	While-Schleife wird durchlaufen, so lange `expression` true ist.

`if expression`
`  (commands)`
`end`
	`if-then`–Konstruktion.

`if expression`
`  (commands1)`
`else`
`  (commands2)`
`end`
	`if-then-else`–Konstruktion.

`if expression1`
`  (commands1)`
`elseif expression2`
`  (commands2)`
`else`
`  (commands3)`
`end`
	`if-then-elseif-else`–Konstruktion.

`switch expression`
`  case test_expresssion1`
`  (commands1)`
`  case {test_expression1`
`  test_expression2,`
`  test_expression3}`
`  (commands2)`
`  otherwise`
`  (commands3)`
`end`
	Case–Konstruktion. Es wird der erste zutreffende Case–Block ausgeführt. Nicht mehrere, wie zum Beispiel in C.

`break`	Sofortiger Sprung aus `while`– oder `for`–Schleife.
`continue`	Sofortiger Sprung zum `end` der `while`– oder `for`–Schleife. Die Schleife wird dann fortgesetzt.

9.3 Regeln

- Funktionsname und Name des m–Files sollen gleich sein.
- Für Funktionsnamen gelten die gleichen Regeln wie für Variablennamen.
- Um Kompatibilität zwischen Plattformen zu erreichen, sollten Funktionsnamen klein geschrieben werden.
- Erste Zeile enthält `function` mit Name der Funktion und Aufrufsyntax. Alle Parameter (input und output) sind lokale Variable. Es ist nicht möglich, Werte über die Input–Variablen zurückzuliefern (call–by–reference).
- Die ersten zusammenhängenden Kommentar–Zeilen nach der Funktionsdeklaration bilden den Help–Text der Funktion. Die erste Zeile (H1) wird mit dem Kommando `lookfor` durchsucht.
- Die Funktion wird beendet, wenn alle Zeilen des m–File abgearbeitet sind, oder wenn `return` aufgerufen wird.
- Vorzeitiger Abbruch der Funktion und Rücksprung zum Command–Window mit `error`. Beispiel:

```
if length(x) > 1
   error('x muss ein skalarer Wert sein.')
end
```

- Warnungen werden an das Command–Window geschickt mit `warning`. Aufruf wie bei `error`, aber die Ausführung der Funnktion wird fortgeführt.

  Warnungen können mit `warning on` und `warning off` ein– oder ausgeschaltet werden.
- Funktionen können auch Skript–m–Files aufrufen. In diesem Fall greift das Skript–m–File auf die lokalen Variablen der Funktion zu, nicht auf die globalen Variablen.

9.4 Funktions–Parameter und Function–Workspace

Parameter

- Funktionen können keine Input– und keine Output–Parameter haben.
- Funktionen können mit weniger Argumenten Aufgerufen werden, als vorgesehen sind. Sie können nicht mit mehr Argumenten aufgerufen werden, als vorgesehen sind.
- Die Anzahl der Input–Parameter ist mit der Funktion `nargin` verfügbar, die Anzahl der Output–Argumente mit `nargout`.
- Beim Aufruf einer Funktion werden die Input–Variablen nicht kopiert, sondern nur lesbar gemacht. Solange sie nicht geändert werden, findet kein Kopiervorgang statt. Vorsicht: `function x = filter(x)` kopiert die Variable `x` (Performance!).

Function–Workspace

- Die Variablen innerhalb einer Funktion sind lokal.
- Mit `global varname` werden Variablen deklariert, auf die innerhalb von anderen Funktionen, oder vom Workspace aus zugegriffen werden kann. Dies ist auch bei rekursiven Funktionsaufrufen sinnvoll sein.

9.5 Function Handles, FEVAL, etc.

Function Handles, FEVAL, EVAL EVALIN

`@fname`	Handle auf Funktion `fname`.
`feval(f,x1,...,xN)`	Die Funktion `f` wird mit den Argumenten `x1,...,xN` aufgerufen. `f` ist entweder ein String mit dem Namen der Funktion oder ein Handle auf eine Funktion.
`[x1,...,xN] = feval(f,x1,...,xN)`	Die Rückgabewerte der Funktion werden in den Variablen gespeichert.
`eval(s)`	Der String `s` wird als Kommando aufgefasst und ausgeführt.
`[x1,...,xN] = eval(s)`	.Die Rückgabewerte des Kommandos in `s` werden in den Variablen gespeichert.
`evalin(WS,s)`	Wie `eval`. Allerdings wird das Kommando im Workspace `WS` ausgeführt. Für WS kann `'caller'` (der Workspace der aufrufenden Funktion) oder `'base'` (der Workspace des Command Windows) verwendet werden.
`assignin(WS,'name',v)`	.Der Variablen `name` wird der Wert `v` zugewiesen. Die Variable wird im Workspace `WS` definiert. `WS` kann `'caller'` oder `'base'` sein.
`inputname`	Liefert innerhalb einer Funktion die Namen der Variablen, mit der die Funktion aufgrufen wurde: `inputname(1)`, `inputname(2)`, etc.

10 Strings

10.1 Einführung

- Strings — genauer Character Strings — sind in Matlab Zeilenvektoren, deren Einträge Zeichen im ASCII–Format sind.
- Daher funktionieren alle Array–Funktionen auch für Strings (Zugriff auf Buchstaben, Substrings, etc.).
- Sollen Strings (Zeilenvektoren) untereinander angeordnet werden, ist darauf zu achten, dass die Strings gleiche Länge haben.
- Daher gibt es spezielle Funktionen, um Strings mit Blanks aufzufüllen (zum Beispiel `char`) und um Blanks zu entfernen (`deblank`).
- Wichtig ist die Funktion `eval`, mit der ein String als Kommando interpretiert wird.

10.2 String–Funktionen

String–Funktionen

`blanks(n)`	String mit n Blanks.
`char(S1,S2,...)`	Vertikale Anordnung der Strings in einem Array.
`strcat(S1,S2,...)`	Horizontales Aneinanderfügen von Strings. Funktioniert auch für Array. Dabei werden Blanks gelöscht.
`strvcat(S1,S2,...)`	Wie `char`, aber Leerzeilen werden ignoriert.
`ischar(S)`	True für einen String–Array.
`isletter(S)`	True für Buchstaben.
`isspace(S)`	True für Whitespaces.
`strcmp(S1,S2)`	Wahr, falls Strings gleich sind.
`strcmp(S1,S2,n)`	Wahr, falls erste n Zeichen der Strings gleich sind.
`strcmpi(S1,S2)`	Wie `strcmp`, aber Groß–Klein–Schreibung wird ignoriert.
`strcmpi(S1,S2,n)`	Wie `strcmp`, aber Groß–Klein–Schreibung wird ignoriert.
`findstr(S1,S2)`	.Finde einen String in dem anderen.
`strmatch(S1,S2)`	Sucht im Stringarray `S2` die Strings, die mir `S1` beginnen, bzw. die, die exakt identisch mit `S1` sind.
`strtok(S1)`	Liefert erstes Token in `S1`, das durch einen Whitespace getrennt ist.
`strtok(S1,T)`	Wie `strtok(S1)`, aber mit `T` statt den Whitespaces.

`double(S)`	Konvertierung des Strings in seine ASCII–Darstellung.
`num2str`	Konvertierung Zahl in String.
`int2str`	Konvertierung Integer in String.
`mat2str`	Konvertierung Matrix in String. Gut für anschließende String–Evaluation.
`str2num`	Konvertierung String–Array in numerisches Array.
`str2double`	Konvertierung String in Double.
`deblanks(S)`	Blanks am Ende des Strings entfernen.
`upper(S)`	Konvertierung zu Großbuchstaben.
`lower(S)`	Konvertierung zu Kleinbuchstaben.
`strrep(S1,S2,S3)`	Ersetzen von `S2` in `S1` mit `S3`.
`strjust(S1,type)`	Ausrichtung des Strings. `type`: `'left'`, `'right'` oder `'center'`.
`eval(S)`	Ausführen des Strings als Kommando.
`T = evalc(S)`	Wie `eval`, aber das Resultat wird im String `T` gespeichert.
`sprintf(S)`	Erzeuge String mit Formatier-Anweisungen (wie in C).
`sscanf(S)`	Lese String mit Formatier-Anweisungen (wie in C).

11 Input/Output

11.1 Load und Save

- Variablen des Workspace können mit `load` und `save` geladen und gespeichert werden.
- Die Files sind vom Typ `.mat`.
- Internes (aber dokumentiertes) File–Format.

Load und Save

`save`	Speichert alle Variablen im File `matlab.mat`.
`save fname var1 var2`	Speichert `var1` und `var2` im File `fname.mat`.
`save -ascii fname.ext var`	Speichert die Variable `var` (typischerweise ein Array) im ASCII–File `fname.ext`.
`load`	Liest alle Variablen aus dem File `matlab.mat`.
`load fname var1 var2`	Liest die Variablen `var1` und `var2` aus dem File `fname.mat`.
`load('fname','var1','var2')`	Wie oben..
`x = load('fname','var1','var2')`	Liest die Variablen `var1` und `var2` aus dem File `fname.mat` und speichert sie als `x.var1` und `x.var2` im Workspace.
`exist('fname.mat','file')`	Liefert den Rückgabewert 2, falls das File `fname.mat` existiert, und 0, falls das File nicht exisitiert.
`whos -file fname.mat`	Listet alle Variablen, die im File `fname.mat` gespeichert sind.
`delete fname.ext`	Löscht das File `fname.ext`.

11.2 Import/Export

Import/Export

`csvread`	Liest „comma seperated values“, z.B. exportiert aus Excel. Funktioniert nur für numerische Daten. Auch Bereiche können angegeben werden.
`csvwrite`	Analog zum Schreiben von csv–File. Diese können z.B. in Excel importiert werden.
`dlmread`	Liest ASCII-Files, in denen die Zahlenwerte durch ein besonderes Zeichen (delimiter) getrennt sind, z.B. Blank, Strichpunkt, Tabulator, oder jedes andere beliebige Zeichen. Es ist auch möglich, nur Teile einer Tabelle einzulesen.
`dlmwrite`	Schreibt entsprechende Files.
`textread`	Kann Files mit verschiedenen Datentypen lesen. Zeilenweise und durch whitespaces getrennt. Datentypen pro Spalte müssen einheitlich sein und werden im Kommando eingegeben (ähnlich wie bei `printf` in C).
`textwrite`	Schreibt entsprechende Files.
`help fileformats`	Informationen über die unterstützten File–Formate.

11.3 Low–Level–I/O

Low–Level–I/O

`fopen`	Öffnet File.
`fclose`	Schließt File.
`fread`	Lesen eines Blockes aus binärem File.
`fwrite`	Schreiben eines Blockes in binäres File.
`fscanf`	Lesen von formatierten ASCII–Daten.
`fprintf`	Formatiertes Schreiben von ASCII–Daten.
`fgetl`	Einlesen einer Zeile (ohne Newline–Character).
`fgets`	Einlesen einer Zeile (mit Newline–Character).
`ferror`	Liefert File–Status.
`feof`	Test für EOF.
`fseek`	Ändere File Position Pointer.
`ftell`	Lese File Position Pointer.
`frewind`	„Zurückspulen“.

11.4 Formatierte Ausgabe auf der Konsole

- `fprintf` ohne Filepointer bewirkt eine formatierte Ausgabe auf der Konsole.

12 Polynome

12.1 Konstruktion und Auswertung

- Ein Polynom
$$p(x) = 4x^3 - 2x + 1$$
wird in Matlab durch den Vektor der Koeffizienten dargestellt:

```
p = [ 4 0 -2 1]
```

Polynome — Konstruktion und Auswertung

`p = [ 3 -2 0]`	Konstruktion des Polynoms $p(x) = 3x^2 - 2x$-
`roots(p)`	Die Nullstellen des Polynoms `p`.
`poly(r)`	Polynom, dessen Nullstellen durch `r` definiert sind (inklusive Vielfachheit). Leitkoeffizient ist 1.
`polyval(p,x)`	Auswertung von $p(x)$.

12.2 Operationen

Polynome — Operationen

`conv(p,q)`	Konvolution der Vektoren `p` und `q`. Das heißt: Multiplikation der Polynome, die durch `p` und `q` repräsentiert werden.
`[p,r] = deconv(a,b)`	Polynomdivision der Polynome `a` und `b`. Das Ergebnis ist das Polynom `p` und der Rest `r`.
`polyder(p)`	Ableitung des Polynoms `p`.
`polyint(p)`	Integration des Polynoms `p`.
`polyint(p,C)`	Integration des Polynoms `p`. Integrationskonstante ist `C`.

12.3 Lineare Regression

Lineare Regression

`p = polyfit(x,y,n)`	Polynom p der Ordnung `n`, das die Datenpunkte (x_i, y_i) im Sinne kleinster Fehlerquadrate optimal approximiert.

13 Daten–Analyse

13.1 Elementare Daten–Analyse

Elementare Daten–Analyse

Die folgenden Befehle arbeiten bei Vektoren spalten– bzw. zeilenweise.
Bei Arrays arbeiten die folgenden Befehle per default spaltenweise. Eine spezielle Aufruf–Syntax erlaubt die Wirkung der Befehle in anderen Dimensionen (`dim`).

`max(x)`	Maximum der jeweiligen Spalten.
`max(x,[],dim)`	Maximum entlang der Dimension `dim`.
`min(x)`	Minimum der jeweiligen Spalten.
`min(x,[],dim)`	Minimum entlang der Dimension `dim`.
`sum(x)`	Summe der jeweiligen Spalten–Einträge.
`sum(x,dim)`	Summe der Einträge entlang der Dimension `dim`.
`prod(x)`	Produkt der jeweiligen Spalten–Einträge.
`prod(x,dim)`	Produkt der Einträge entlang der Dimension `dim`.
`diff(x)`	Differenz aufeinanderfolgender Einträge der jeweiligen Spalten.
`diff(x,1,dim)`	Differenz aufeinanderfolgender Einträge entlang der Dimension `dim`.
`cumsum(x)`	Kummulierte Summe der jeweiligen Spalten–Einträge.
`cumsum(x,dim)`	Kummulierte Summe der Einträge entlang der Dimension `dim`.

13.2 Elementare statistische Daten–Analyse

Elementare statistische Daten–Analyse

Die folgenden Befehle arbeiten bei Vektoren spalten– bzw. zeilenweise.
Bei Arrays arbeiten die folgenden Befehle per default spaltenweise. Eine spezielle Aufruf–Syntax erlaubt die Wirkung der Befehle in anderen Dimensionen (`dim`).

`mean(x)`	Mittelwert der jeweiligen Spalten.
`mean(x,dim)`	Mittelwert entlang der Dimension `dim`.
`median(x)`	Median der jeweiligen Spalten.
`median(x,dim)`	Median entlang der Dimension `dim`.
`std(x)` oder `std(x,0)`	Standardabweichung (erwartungstreuer Schätzer): $\sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i-\bar{x})^2}$.
`std(x,1)`	Standardabweichung: $\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i-\bar{x})^2}$.
`std(x,flag,dim)`	Standardabweichung entlang der Dimension `dim`.
`corrcoef(x,y)`	Korrelationskoeffizient der Daten `x` und `y`.

13.3 Interpolation

Interpolation

`yi = interp1(x,y,xi,'method')`	Interpoliert die Funktion, die durch die Datenvektoren `x` und `y` definiert ist, und wertet sie an der Stelle bzw. den Stellen `xi` aus. Verschiedene Methoden stehen zur Verfügung:
`'nearest'`	Nearest–Neighbour–Interpolation.
`'linear'`	Stückweise lineare Interpolation.
`'spline'`	Interpolation mit kubischen Splines.
`'cubic'`	Kubische Interpolation (Monotonie–erhaltend).

14 Logische Funktionen

Logische Funktionen

`ispc`	Wahr, falls auf PC ausgeführt.
`isunix`	Wahr, falls unter UNIX ausgeführt.
`isglobal`	Wahr, falls die Variable global ist.
`isempty`	Wahr, falls Array leer ist.
`isequal`	Vergleich für beliebige Datentypen.
`isfinite`	Wahr falls Zahl nicht `Inf`, `-Inf` oder `nan` ist.
`isinf`	Wahr, falls Zahl `Inf` oder `-Inf` ist.
`islogical`	Wahr für logisches Array.
`isnan`	Wahr falls Zahl `nan` ist.
`isnumeric`	Wahr, falls Variable eine Zahl darstellt.
`isreal`	Wahr für Zahlen ohne Imaginärteil.
`isprime`	Wahr für Primzahlen.
`inpolygon`	Inside–Polygon–Test.
`isvarname`	Wahr, falls String ein gültiger Variablenname ist.
`iskeyword`	Wahr, falls String ein reserviertes Keyword ist.
`issparse`	Wahr, falls die Variable eine „sparse matrix“ repräsentiert.
`ishandle`	Wahr, falls Variable ein Handle auf ein Grafikobjekt ist.
`ischar`	Wahr für ein Character String Array.
`isletter`	Wahr für Buchstaben.
`isspace`	Wahr für Whitespaces.

EXIST

`exist('A')` liefert den Rückgabewert

0	falls `A` nicht existiert.
1	falls `A` eine Variable im Workspace ist.
2	falls `A` ein File im Suchpfad ist.
7	falls `A` ein Verzeichnis ist.

```
help exist
```