Kapitel 9

Arrays (Felder)

Arrays

- Bisher wurden nur Variablen verwendet, die einen einzelnen Wert aufnehmen können.
- Diese Variablen werden auch Skalare genannt.
- Aus einer Menge von Skalaren gleichen Datentyps kann man ein Feld (Array) bilden.
- Die einzelnen Skalare werden dann durch Indizes unterschieden.
- Standardmäßig sind in Matlab alle Variablen Arrays; meist eben nur mit einem Element.

Arrays

- In der Mathematik werden Arrays z.B. für die Darstellung von Vektoren und Matrizen verwendet.

$$v = (v_1, v_2, ..., v_n)$$

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

- Vektoren sind eindimensionale und Matrizen zweidimensionale Arrays.

Definition von Arrays

- Das Array wird definiert durch die [].
- Jedes Element des Arrays hat den gleichen Datentyp.
- Werden in ein Array unterschiedliche Datentypen geschrieben, so wird versucht, alle auf einen gemeinsamen Datentyp zu bringen.
- Der Zugriff auf die einzelne Elemente geschieht mittels Index.

Eindimensionale Arrays

- Im folgenden Beispiel wird ein eindimensionales Array mit dem Namen *a* und dem Inhalt 1, 2, 3, 4 und 5 definiert.
- Das Array wird definiert durch die [].
- Das Array hat 5 Elemente.
- Alle Elemente sind vom Datentyp *int*.

```
a = [1 2 3 4 5]
```

1	2	3	4	5

Eindimensionale Arrays

- Im folgenden Beispiel wird ein eindimensionales Array mit dem Namen *werte* und dem Inhalt 1, 2, 3, 4 und "hallo" definiert.
- Das Array hat 5 Elemente.
- Alle Elemente sind vom Datentyp *string*.

```
werte = [1 2 3 4 "hallo"]
```

"1"	"2"	"3"	"4"	"hallo"

Man sieht, dass der Datentyp angeglichen wird. Der gemeinsame Datentyp aller Werte ist: *string*

Eindimensionale Arrays

- Im folgenden Beispiel wird ein eindimensionales Array mit dem Namen *komma* und dem Inhalt 1, 2, 3, 4 und 5.2 definiert.
- Das Array hat 5 Elemente.
- Alle Elemente sind vom Datentyp *double*.

```
komma = [1 2 3 4 5.2]
```

```
1.0000    2.0000    3.0000    4.0000    5.2000
```

Man sieht, dass der Datentyp angeglichen wird. Der gemeinsame Datentyp aller Werte ist: *double*

Eindimensionale Arrays

- Man kann beim Erstellen eines Arrays die Elemente entweder mit einem Leerzeichen

```
werte = [6 7 8 9 10];
```

oder mit einem Komma

```
werte = [6,7,8,9,10];
```

trennen und erhält somit einen Ein-Zeilen-Vektor mit x Spalten.

Eindimensionale Arrays

- Zugriff auf einzelne Elemente hat man, indem man das Element durch seinen Index (Position) anspricht.

```
a = [6 7 8 9 10];
disp(a(2));
```

```
7
```

Eindimensionale Arrays

- Der Zugriff auf ein Element, welches vorher **nicht** definiert wurde, resultiert in einer Fehlermeldung von Matlab.

```
a = [6 7 8 9 10];
disp(a(6));
```

```
Index exceeds the number of array elements (5).
```

Initialisierung von eindimensionalen Arrays

- Wenn man die Größe eines Arrays, aber noch nicht den Inhalt, also dessen Elemente kennt, dann sollte man das Array trotzdem initialisieren (mit einem Startwert belegt).

```
a = ones(1, 5);                    % 1 Zeile, 5 Spalten
```

- Dies erstellt ein Array mit folgendem Inhalt:

```
1   1   1   1   1
```

Initialisierung von eindimensionalen Arrays

- Zur Initialisierung eines Arrays sind diese Befehle von Matlab sehr hilfreich:

zeros	Erzeugt ein Array bei dem alle Elementen 0 sind.
ones	Erzeugt ein Array bei dem alle Elementen 1 sind.
rand	Erzeugt ein Array bei dem alle Elementen eine Zufallszahl zwischen 0 und 1 sind.

Initialisierung von eindimensionalen Arrays

- Hier ein paar einfache Beispiele. Die Befehle bieten darüber hinaus noch viel mehr Möglichkeiten. (siehe Matlab-Hilfe)

zeros(1, 3)	Erzeugt ein Array mit: 1 Zeile und 3 Spalten bei dem alle Elemente 0 sind.
ones(2, 4, 3)	Erzeugt ein Array mit: 2 Zeilen, 4 Spalten und 3 Ebenen bei dem alle Elemente 1 sind.
rand(4)	Erzeugt ein Array mit: 4 Zeilen und 4 Spalten bei dem alle Elemente Zufallszahlen zwischen 0 und 1 sind.

Initialisierung von eindimensionalen Arrays

- Das Erstellen und Initialisieren von Arrays wirkt sich auf die Geschwindigkeit aus.

```
x = ones(1, 10);
for i = 1:10
    x(i) = input("Wert eingeben");
end
```

```
for i = 1:10
    x(i) = input("Wert eingeben");
end
```

Initialisierung von eindimensionalen Arrays

- Auch, wenn der Unterschied nicht ins Gewicht fällt, ist das linke Programm bei der Verarbeitung der Programmier-Befehle schneller.

- Das linke Programm hat sich durch die Initialisierung schon Speicher für die zu erwartenden 10 Elemente reserviert. (Und diesen Speicher mit Einsen gefüllt.)
  In der Schleife wird dann nur noch die 1 durch den eingegebenen Wert ersetzt.

Initialisierung von eindimensionalen Arrays

- Das rechte Programm hingegen muss bei jedem Schleifendurchlauf das Array x um ein Element erweitern. Dazu muss Matlab für das neue, größere Array Speicher reservieren, die alten Werte dorthin kopieren und dann das alte Array aus dem Speicher entfernen. Dies dauert effektiv länger, als das linke Programm.

Ändern der Größe eines Arrays

- In Matlab hat ein Array keine festgelegte Größe.
- Es kann bei Bedarf erweitert oder auch verringert werden.
- Um dies zu erreichen spricht man durch den Index die gewünschte Position an.
- Dort kann man ein Element entweder neu setzen oder das dort vorhandene Element löschen.

Hinzufügen von Elementen zu einem Array

- Im folgenden Beispiel haben wir ein Array mit 4 Elementen.
- Wir fügen nun ein neues Element an der 5. Position hinzu. Dies ist soweit kein Problem, da das neue Element gleich einen Wert mitbringt und dieser gesetzt wird.
- Fügt man nun noch ein Element an Position 10 hinzu, obwohl das Array momentan nur 5 Elemente hat, so werden die dazwischen liegenden neuen Elemente mit 0 (bei Zahlen-Array) bzw. mit <missing> (bei string-Array) aufgefüllt.

Hinzufügen von Elementen zu einem Array

```
z = [1 2 3 4];
disp(z);

z(5) = 7;          % 5. Element hinzufügen
disp(z);

z(10) = 8;         % 10. Element hinzufügen; Dabei werden alle dazwischen
disp(z);           % liegenden, neuen Elemente mit dem Wert 0 bei
                   % Zahlen oder mit dem Wert <missing> bei strings belegt.
```

```
1   2   3   4
1   2   3   4   7
1   2   3   4   7   0   0   0   0   8
```

Löschen von Elementen aus einem Array

- Im folgenden Beispiel haben wir ein Array mit 7 Elementen.
- Wir löschen das Element an der 7. Position.
  Da dies sowieso das letzte Element war, muss hier nichts weiter beachtet werden.
- Wir löschen nun die Elemente von Position 2 bis Position 5.
  Hierbei werden die dahinter liegenden Elemente entsprechend verschoben.

Löschen von Elementen aus einem Array

```
z = [1 2 3 4 5 6 7];
disp(z);

z(7) = [];              % 7. Element löschen
disp(z);

z(2:5) = [];            % 2. bis 5. Element löschen; Dabei werden alle
disp(z);                % dahinter liegenden Elemente nach links
                        % verschoben.
```

```
1   2   3   4   5   6   7
1   2   3   4   5   6
1   6
```

Kopieren von eindimensionalen Arrays

- Das Kopieren eines Arrays erfolgt durch eine einfache Zuweisung.

```
z = [1 2 3 4 5];
a = z;
disp(a);
```

```
1    2    3    4    5
```

Die Größe eines eindimensionalen Arrays

- Der Befehl *size()* ermittelt die Größe des Arrays.
- Angegeben werden dabei die einzelnen Dimensionen.

```
z = [1 2 3 4 5];
[m,n] = size(z);
fprintf("Zeilen: %i\n", m);
fprintf("Spalten: %i\n", n);
```

```
Zeilen: 1
Spalten: 5
```

Eindimensionales Array - Beispiel

- In folgendem Beispiel wird der Benutzer zur Eingabe von 10 Zahlen aufgefordert.
- Danach werden die Zahlen sortiert und wieder ausgegeben.
- Siehe Beispiel 9 (auf der Webseite zu finden).

Mehrdimensionale Arrays

- Das Maximum an Dimensionen eines Arrays wird nur begrenzt durch das System (32/64 Bit, RAM, etc).
- Die am häufigsten verwendeten mehrdimensionalen Arrays sind zweidimensionale Arrays.
- Die Zeilen werden durch ein Semikolon unterteilt.

```
a = [1 2 3; 4 5 6; 7 8 9]     % Array mit 2 Dimensionen (Matrix)
```

```
1   2   3
4   5   6
7   8   9
```

Mehrdimensionale Arrays

- Zugriff auf einzelne Elemente hat man, indem man das Element durch seinen Index (Position) anspricht.
- Durch Zeile und Spalte:

```
a = [1 2 3; 4 5 6; 7 8 9]
disp(a(2, 3));
```

```
6
```

Mehrdimensionale Arrays

- Oder durch einen linearen Index, der von einer spaltenweisen Nummerierung der Elemente ausgeht:

```
a = [11 12 13; 14 15 16; 17 18 19]
disp(a(6));
```

```
18
```

11	12	13
14	15	16
17	18	19

Initialisierung von mehrdimensionalen Arrays

- Auch hier kann man wieder die Befehle zeros, ones, rand benutzen.

```
a = zeros(3, 3)                    % 3 Zeilen, 3 Spalten
```

- Dies erstellt ein Array mit folgendem Inhalt:

```
0    0    0
0    0    0
0    0    0
```

Initialisierung von mehrdimensionalen Arrays

- Man kann bei der Initialisierung auch den Doppelpunkt benutzen.

```
a = [1:2:5; 3 2 1; 10:12]
```

- Dies erstellt ein Array mit folgendem Inhalt:

```
 1   3   5
 3   2   1
10  11  12
```

Zuweisung von Spalten / Zeilen

- Man kann auch ganze Zeilen oder Spalten zuweisen.

```
a = [1:2:5; 3 2 1; 10:12]
z = a(:, 2);    % alle Zeilen der 2. Spalte
```

```
 1   3   5
 3   2   1
10  11  12
```

```
 3
 2
11
```

Kopieren von mehrdimensionalen Arrays

- Das Kopieren eines mehrdimensionalen Arrays erfolgt (wie auch bei eindimensionalen Arrays) durch eine einfache Zuweisung.

```
z = [1 2 3; 4 5 6; 7 8 9]
a = z;
```

Die Größe eines mehrdimensionalen Arrays

- Angegeben werden dabei die einzelnen Dimensionen.

```
z = [1 2 3; 4 5 6; 7 8 9];
[m, n, o] = size(z);
disp(m);
disp(n);
disp(o);
```

```
3
3
1
```

Mehrdimensionale Arrays - Beispiel

- In folgendem Beispiel werden eine 3x3-Matrix und ein Vektor mit 3 Elementen definiert.
- Es wird der Vektor in die zweite Zeile der Matrix kopiert und die Matrix dann auf den Bildschirm ausgegeben.
- Siehe Beispiel 10 (auf der Webseite zu finden).

Mehrdimensionale Arrays

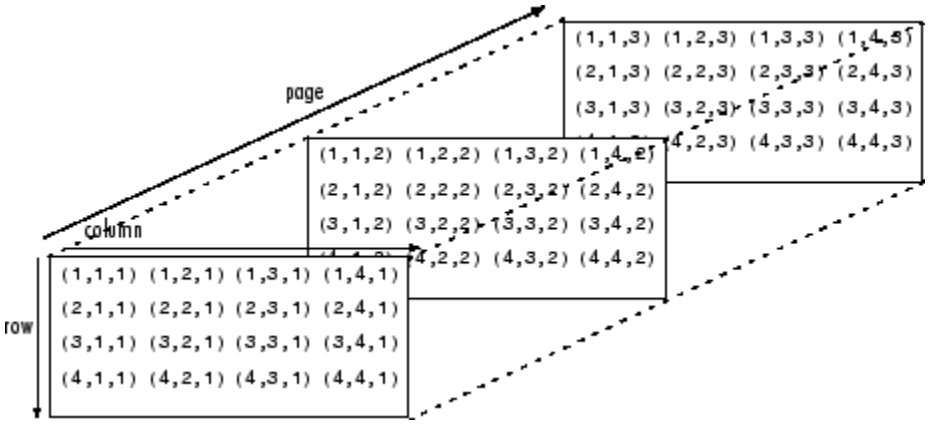
- Ein array mit 3 Dimensionen.
- Man erstellt erst ein Array mit zwei Dimensionen.

```
a = [1 2 3; 4 5 6; 7 8 9]
```

```
1   2   3
4   5   6
7   8   9
```

Mehrdimensionale Arrays

- Dieses Array ist quasi Seite 1 des 3-dimensionalen Arrays



(Bild entnommen aus der Matlab-Online-Hilfe: https://de.mathworks.com/help/matlab/math/multidimensional-arrays.html)

Mehrdimensionale Arrays

- Nun kann man weitere Seiten (sozusagen als Erweiterung der 3. Dimension) hinzufügen.

```
a(:, :, 2) = [10 11 12; 13 14 15; 16 17 18]
```

- Durch die beiden Doppelpunkte werden die Zahlen auf der rechten Seite als Zeilen und Spalten übernommen.
- Die 2 bezeichnet im 3-dimensionalen Array die 2. Seite.

Mehrdimensionale Arrays

- Man kann die Seiten auch einzeln definieren und anschließend zusammenhängen.

```
Seite_1 = [1 2 3; 4 5 6; 7 8 9];
Seite_2 = [10 11 12; 13 14 15; 16 17 18];
Seite_3 = [20 21 22; 23 24 25; 26 27 28];
a_3D = cat(3, Seite_1, Seite_2, Seite_3);
disp(a_3D(1, 1, 2));
```

```
10
```

Cell Arrays

- Möchte man in einem Array unterschiedliche Datentypen speichern, so nutzt man ein Cell Array.

```
a = {1 2 3; 'Test' 5 6; 7 8 [9 10]}
```

```
{[   1]}   {[2]}   {[            3]}
{'Test'}   {[5]}   {[            6]}
{[   7]}   {[8]}   {[1x2 double]}
```

Ausgabe bei Cell Arrays

- Zur Information: Gibt man das Cell Array an ohne abschließendes Semikolon, so hat man eine etwas andere Ausgabe, als wenn man zur Ausgabe *disp()* nutzt.

```
a = {1 2 3; 'Test' 5 6; 7 8 [9 10]}
```

```
{[   1]}   {[2]}   {[          3]}
{'Test'}   {[5]}   {[          6]}
{[   7]}   {[8]}   {[1x2 double]}
```

Ausgabe bei Cell Arrays

- Bei *disp()* hingegen:

```
a = {1 2 3; 'Test' 5 6; 7 8 [9 10]};
disp(a);
```

```
[   1]    [2]    [           3]
'Test'    [5]    [           6]
[   7]    [8]    [1x2 double]
```

(Hinweis: Aufgrund der besseren Lesbarkeit wird im Skript die Ausgabe von Cell Arrays durch disp() benutzt.)

Initialisierung von Cell Arrays

- Die Initialisierung eines Cell Arrays geschieht durch den Befehl *cell*.

```
a = cell(2, 2);
```

```
[]    []
[]    []
```

Initialisierung von Cell Arrays

- Sind die Werte schon zu Beginn bekannt, kann man das cell array auch sofort bei Erstellung füllen. Dies geschieht durch die geschweiften Klammern {}.
- Hier wird pro Zelle eine Zahl gespeichert.

```
a = {[1], [2]; [3], [4]};
```

oder

```
a = {1, 2; 3, 4};
```

```
[1]   [2]
[3]   [4]
```

Initialisierung von Cell Arrays

- Man kann auch pro Zelle komplexere Daten speichern. Zum Beispiel eine ganze Matrix.
- Hier wird pro Zelle eine Matrix gespeichert. Wir haben also in unserem Cell Array 2 Matrizen gespeichert.

```
a = {[1, 2; 3, 4], [5, 6; 7, 8]};
```

```
[2x2 double]    [2x2 double]
```

Ausgabe aller Elemente eines Cell Arrays

- Die Ausgabe geschieht durch *celldisp()*. Dadurch werden alle Zellen ausgegeben.

```
a = {[1, 2; 3, 4], [5, 6; 7, 8]};
celldisp(a);
```

```
a{1} = [1]   [2]
       [3]   [4]
a{2} = [5]   [6]
       [7]   [8]
```

Ausgabe eines Elementes eines Cell Arrays

- Die Ausgabe ist möglich durch den Befehl *disp()*. Man muss hierbei angeben, welche Zelle ausgegeben werden soll.

```
a = {[1, 2; 3, 4], [5, 6; 7, 8]};
disp(a{2});
```

```
a{2} = [5]    [6]
       [7]    [8]
```

Ausgabe eines Wertes einer Matrix einer Zelle eines Cell Arrays

- Beinhaltet die Zelle eine Matrix, von der man nur einen Wert ausgeben möchte, benötigt man bei *disp()* zusätzlich zur Zelle noch die Angabe welche Position in der Zelle ausgegeben werden soll.

```
a = {[1, 2; 3, 4], [5, 6; 7, 8]};
disp(a{2}(1,2));
```

```
6
```

Ausgabe Cell Arrays - allgemein

- a() = gibt die entsprechende Zelle zurück
- a{} = gibt den Inhalt der entsprechenden Zelle zurück

```
a = {[1, 2; 3, 4], [5, 6; 7, 8]};
disp(a(2));
disp(a{2});
disp(a{2}(1,2));
```

```
[2x2 double]
5   6
7   8
6
```

Hinzufügen von Elementen in ein Cell Array

- Das vorher initialisierte und damit erzeugte, leere Cell Array kann durch Werte beschrieben werden, indem man den Index der Zelle angibt.

```
a = cell(2, 2);
a{1,2} = 2
```

```
[]   [2]
[]    []
```

Löschen von Elementen aus einem Cell Array

```
a = {[1, 2; 3, 4], [5, 6; 7, 8], [9, 10; 11, 12]};
disp(a);

a(2) = [];              %2. Zelle des cell arrays löschen
disp(a);

a{2} = [];              % Inhalt der 2. Zelle des cell arrays löschen
disp(a);
```

```
[2x2 double]    [2x2 double]    [2x2 double]
[2x2 double]    [2x2 double]
[2x2 double]    []
```

Sortierung von Elementen in einem Cell Array

- Eine Besonderheit von Cell Arrays ist, dass sie sich relativ einfach sortieren lassen.
- Durch den Befehl *sortrows()* kann man zudem angeben, nach welcher Spalte die Zeilen sortiert werden sollen.

Sortierung von Elementen in einem Cell Array

```
charMatrix = {'c', 'x'; 'a', 'z'; 'd', 'w'; 'b', 'y'};
```

unsortiert

'c'	'x'
'a'	'z'
'd'	'w'
'b'	'y'

sortrows(charMatrix)

'a'	'z'
'b'	'y'
'c'	'x'
'd'	'w'

sortrows(charMatrix,2)

'd'	'w'
'c'	'x'
'b'	'y'
'a'	'z'

Sortierung von Elementen in einem Cell Array

```
charMatrix = {'c', 'x'; 'a', 'z'; 'd', 'w'; 'b', 'y'};
disp(charMatrix);
sortiert = sortrows(charMatrix);
disp(sortiert);
sortiert = sortrows(charMatrix, 2);
disp(sortiert);
```

Structure Array

- Eine andere Möglichkeit, Daten zu speichern sind sogenannte Structure Arrays.
- Jedes Element des Structure Arrays hat den selben Aufbau an benannten Feldern. Und jedes Feld ist wiederum ein "container", der einen beliebigen Datentyp speichern kann.

```
patient.name = "Max Mustermann";
patient.geschlecht = "männlich";
patient.alter = 35;
```

Allgemeines zu Structure Array

- Structure Arrays sind lesbarer, als z.B. Cell Arrays
- Sie bringen jedoch einige Einschränkungen mit sich.
  Zum Beispiel kann man sie, im Gegensatz zu Cell Arrays, nicht so einfach sortieren.

Erstellung eines Structure Array

- Erzeugt wird das Structure Array entweder direkt bei Eingabe der Daten:

```
patient.name = "Max Mustermann";
patient.geschlecht = "männlich";
patient.alter = 35;
disp(patient);
```

```
        name: "Max Mustermann"
geschlecht: "männlich"
         alter: 35
```

Erstellung eines Structure Array

- Oder gleich zu Beginn mit allen Feldnamen:

```
patient = struct('name', 'Max Mustermann', 'geschlecht', 'männlich', 'alter', 35);
disp(patient);
```

```
      name: "Max Mustermann"
geschlecht: "männlich"
     alter: 35
```

Hinzufügen weiterer Elemente in ein Structure Array

- Ein neues Element hinzuzufügen ist ganz einfach:

```
patient(3).name = "Erika Mustermann";
patient(3).geschlecht = "weiblich";
patient(3).alter = 33;
disp(patient);
```

```
1×3 struct array with fields:
    name
    geschlecht
    alter
```

Hinzufügen weiterer Elemente in ein Structure Array

- patient(2) wurde auch erstellt, aber mit leeren Feldern:

```
disp(patient(2));
```

```
      name: []
geschlecht: []
     alter: []
```

Zugriff auf die Felder des Structure Array

- Möchte man einen speziellen Wert ändern, so geschieht das durch die Punktnotation:

```
patient(2).name = 'Hans Wurst';
disp(patient(2));
```

```
        name: 'Hans Wurst'
geschlecht: []
        alter: []
```

Ausgabe des Structure Array

- Die Ausgabe einzelner Werte erfolgt, wie üblich unter Angabe der Position.

```
disp(patient(1).name);
```

```
'Max Mustermann'
```

Ausgabe des Structure Array

- Man kann auch alle Werte eines Feldes im Array ausgeben.
- Ein:

```
patient.name
```

- liefert alle Elemente des Feldes "name":

```
ans =
        'Max Mustermann'
ans =
        'Hans Wurst'
ans =
        'Erika Mustermann'
```

Ausgabe des Structure Array

- Weiterführender Hinweis:

```
patient.name
```

- liefert in diesem Falle ein Array aller Namen.
- Dies kann *disp()* allerdings so nicht verarbeiten. Daher steht patient.name hier ohne expliziten Ausgabe-Befehl.