

Numerik

Ingenieurinformatik Teil 2, Sommersemester 2026

David Straub

Gliederung

1. **Einführung in Matlab**
2. **Arbeiten mit Arrays**
3. **Funktionen und Kontrollstrukturen**
4. **Analysis** (Polynome, Ableitung, Integration, ...)
5. **Lineare Algebra** (Gleichungssysteme, Eigenwerte, ...)
6. **Differentialgleichungen**
7. **Einführung in Simulink**

2. Arbeiten mit Arrays

Felder – Arrays

Eine $n \times m$ -Matrix (2D-Array) ist der zentrale Datentyp in MATLAB.

	Sp. 1	Sp. 2	Sp. 3	Sp. 4
Zeile 1	1	2	-3	4
Zeile 2	8	9	5	6
Zeile 3	11	12	0	9

- Alle Elemente besitzen den gleichen Datentyp (double)
- Zugriff: $A(\text{zeile}, \text{spalte})$, z. B. $A(2,3) = 5$

Begriffe: Array – Matrix – Vektor – Skalar

Begriff	Dimension	Indexzugriff
Array / Feld	n-dimensional	$A(k_1, k_2, \dots, k_n)$
Matrix	2D ($n \times m$)	$A(\text{zeile}, \text{spalte})$
Zeilenvektor	$1 \times n$	$A(1, k)$ $A(k)$
Spaltenvektor	$n \times 1$	$A(k, 1)$ $A(k)$
Skalar	1×1	$A(1, 1)$ $A(1)$ A

- $\text{size}(A)$ gibt die Dimensionen zurück: $\text{size}(A) \rightarrow [n, m]$, $\text{size}(A, 1) \rightarrow$ Anzahl Zeilen, $\text{size}(A, 2) \rightarrow$ Anzahl Spalten
- $\text{ndims}(A)$ gibt die Anzahl der Dimensionen zurück: Vektor/Matrix $\rightarrow 2$, 3D-Array $\rightarrow 3$, ...

Zwei Bedeutungen von „Matrix“

1. Mathematisches/physikalisches Objekt: - Drehmatrix (2×2 , 3×3), Trägheitsmatrix (3×3) - Koeffizientenmatrix ($n \times m$) für n Gleichungen mit m Unbekannten - Eigenschaften wie Rang, Determinante, Eigenwerte sind physikalisch bedeutsam

2. Datenspeicher: - Zusammenfassung zusammengehöriger Daten (z. B. Messwerte) - Rang, Determinante etc. haben hier keine sinnvolle Bedeutung

Beispiel: Drehmatrix (mathematisch/physikalisch)

Drehung eines 3D-Punktes (x, y, z) um die z-Achse mit Winkel θ (z. B. CAD):

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Für $\theta = 30^\circ$:

$$R_z = \begin{pmatrix} 0.866 & -0.5 & 0 \\ 0.5 & 0.866 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- Mathematische Eigenschaften relevant: $\det(R) = 1$, $R^{-1} = R^T$ (orthogonale Matrix)
- Physikalische Bedeutung: Koordinatentransformation in CAD, Robotik, ...

→ Kapitel Lineare Algebra

Beispiel: Batterie-Messreihe als Matrix

Messungen einer Batterie-Entladung zu n Zeitpunkten:

t [s]	I(t) [A]	U(t) [V]	T(t) [°C]
0	5.0	12.6	25.0
10	5.2	12.3	26.5
20	4.8	12.0	27.8
...	...	...	...

→ $n \times 4$ -Matrix oder $4 \times n$ -Matrix. Alternativ: 4 einzelne Vektoren – aber unpraktisch bei Funktionsübergabe.

Arrays erzeugen – Syntax

Grundelemente: - `[ ]` – Array-Konstruktor - `;` – trennt Zeilen (neue Zeile) - `,` (oder Leerzeichen) – trennt Elemente in einer Zeile

Beispiel:

```
A = [1, 2, 3; 4, 5, 6]    % 2×3-Matrix
```

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

Vektoren erzeugen

Zeilenvektor (1×n):

```
x = [1 4 8]
```

$$x = \begin{pmatrix} 1 & 4 & 8 \end{pmatrix}$$

Spaltenvektor (n×1):

```
y = [2; 5; 9]
```

$$y = \begin{pmatrix} 2 \\ 5 \\ 9 \end{pmatrix}$$

Arrays initialisieren

Befehl	Bedeutung	Python (NumPy)
<code>zeros(m, n)</code>	$m \times n$ -Nullmatrix	<code>np.zeros((m, n))</code>
<code>ones(m, n)</code>	$m \times n$ -Einsmatrix	<code>np.ones((m, n))</code>
<code>eye(n)</code>	$n \times n$ -Einheitsmatrix	<code>np.eye(n)</code>
<code>rand(m, n)</code>	$m \times n$ -Zufallsmatrix (gleichverteilt)	<code>np.random.rand(m, n)</code>

```
A = zeros(3, 4)    % 3×4-Nullmatrix
I = eye(3)          % 3×3-Einheitsmatrix
```

Häufiges Muster: Matrix vorallokieren mit `zeros`, dann befüllen – effizienter als schrittweises Erweitern.

Array-Zugriff und -Manipulation

Zugriff:

```
A(2,3)           % Element Zeile 2, Spalte 3
A(2,:)           % komplette Zeile 2
A(:,3)           % komplette Spalte 3
A(1:2, 2:4)      % Teilmatrix (Zeilen 1-2, Spalten 2-4)
A(end)           % letztes Element
```

Manipulation:

```
A(3,2) = 11      % schreibender Zugriff → erweitert Array bei Bedarf
x(3:7) = 0        % mehrere Elemente auf 0 setzen
x(3:8) = []       % Elemente löschen
A'               % transponieren (bei komplexen: konjugiert!)
A.'              % nur transponieren (ohne Konjugation)
```

Linear Indexing

Jedes Element einer Matrix kann auch über einen **einzelnen Index** angesprochen werden – MATLAB zählt dabei **spaltenweise** von oben nach unten, beginnend bei 1:

$$A = \begin{pmatrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{pmatrix}$$

```
A(2,3)    % Zeile 2, Spalte 3 → 8
A(8)       % Linear Index 8    → 8  (identisch!)
A(end)     % letztes Element   → 12
```

- Nützlich für kompakte Ausdrücke (z. B. `A(end)`, `A(:)` für alle Elemente als Spaltenvektor)
- Vorsicht: kann Code schwerer lesbar machen – besser `(zeile, spalte)` wenn der 2D-Kontext wichtig ist

Array Slicing – Teilbereiche auswählen

Colon-Operator `start:end` bzw. `start:step:end` wählt einen Teilbereich aus:

```

A = magic(5);           % 5×5-Matrix als Beispiel

A(2:4, 1:3)            % Zeilen 2–4, Spalten 1–3 → 3×3-Teilmatrix
A(end-1:end, :)        % letzte 2 Zeilen, alle Spalten
A(1:2:end, :)          % jede zweite Zeile (Schrittweite 2)
A(:, [1, 3, 5])        % Spalten 1, 3 und 5

```

Array Slicing – Vergleich MATLAB vs. Python/NumPy

	MATLAB	Python / NumPy
Indexstart	1	0
Bereichsende	inklusiv: 1:3 → 1,2,3	exklusiv: 0:3 → 0,1,2
Klammern	A(zeile, spalte)	A[zeile, spalte]
Letztes Element	end	-1
Schrittsyntax	start:step:end	start:stop:step
Gesamte Spalte	A(:, k)	A[:, k-1]

Array Slicing – Vergleich MATLAB vs. Python/NumPy

% MATLAB	# Python/NumPy	
A(2, 3)	A[1, 2]	% 0-basiert!
A(1:3, 2:4)	A[0:3, 1:4]	% Ende exklusiv!
A(end, :)	A[-1, :]	
A(1:2:end, :)	A[:, 2, :]	% start weglassen → 0
A(:, 2)	A[:, 1]	

Logische Indizierung

Statt numerischer Indizes kann ein **logischer Vektor** (aus true/false) als Index verwendet werden – nur die Elemente, bei denen der Ausdruck true ist, werden ausgewählt.

```

F = [0, 420, 850, 1240, 830, 0];

mask = F > 800           % [false false true true true false]
F(mask)                  % [850, 1240, 830] – nur Werte > 800

```

```
% Kurzform (inline):
F(F > 800)           % identisch
F(F == max(F))       % Element mit dem Maximalwert
```

Typische Anwendung: Messwerte filtern, Ausreißer entfernen, Schwellwerte prüfen.

Nützliche Array-Funktionen

Funktion	Bedeutung
length(x)	Anzahl der Elemente (längste Dimension)
numel(A)	Gesamtanzahl aller Elemente
sum(x)	Summe aller Elemente
min(x) / max(x)	Minimum / Maximum
mean(x)	Mittelwert
sort(x)	Sortierung (aufsteigend)

Bei Matrizen wirken diese Funktionen standardmäßig **spaltenweise** – `mean(A)` liefert einen Zeilenvektor mit den Spaltenmitteln. Mit `mean(A, 'all')` über alle Elemente.

Übung: Zugriff und Berechnung

Gegeben: `x = [3, 7, 2, 9, 1, 6]`

Schreiben Sie jeweils **zwei verschiedene** MATLAB-Ausdrücke:

- a) Das letzte Element von `x`
- b) `x` in umgekehrter Reihenfolge
- c) Nur die Elemente von `x`, die größer als 5 sind
- d) Den Mittelwert aller Elemente

Vergleichen Sie mit Ihrer Nachbarperson – welche Varianten haben Sie gefunden?

Arrays höherer Dimension

In MATLAB werden höherdimensionale Arrays mit `cat` entlang einer Dimension verkettet:

```
A = [1, 2; 3, 4];           % 2x2-Matrix
B = [5, 6; 7, 8];           % 2x2-Matrix
T = cat(3, A, B);           % 2x2x2-Array (3D)
% T(:,:,1) = A, T(:,:,2) = B
```

Alternativen:

```
T(:,:,1) = A;               % direkte Zuweisung
T(:,:,2) = B;
```

```
T = zeros(2, 3, 4);         % 2x3x4-Array mit Nullen initialisieren
```

Unterschied zu NumPy: keine verschachtelten Klammern erlaubt

Operatoren und Arrays

- Die meisten Operatoren und Funktionen wirken **elementweise** auf Arrays
- Ausnahme: Matrizenmultiplikation $*$ ($\rightarrow$ nächste Folie)

```
>> A = [1,2; 3,4]
>> sin(A)           % Sinus aller Elemente (Bogenmaß)
ans =
    0.8415    0.9093
    0.1411   -0.7568
>> B = 5*A + 2       % B(i,j) = 5*A(i,j) + 2
>> y = sin([0, pi/6, pi/4, pi/3])
```

Operatoren – Dimensionsregeln: Fehlerfall

Elementweise Operationen schlagen fehl, wenn in einer Dimension beide Größen > 1 sind und **nicht übereinstimmen**:

```
>> A = [1,2; 3,4]       % 2x2
>> y = [1; 2; 3]        % 3x1 – 3 Zeilen passen nicht zu 2 Zeilen
>> A + y
Error using +
Matrix dimensions must agree.
```

$\rightarrow$ Typischer Anfängerfehler: Spalten- statt Zeilenvektor, oder transponiert vergessen.

Operatoren – Dimensionsregeln: Implicit Expansion

Seit R2016b expandiert MATLAB Arrays mit **kompatiblen** Dimensionen automatisch:

Kompatibel = in jeder Dimension gleich groß **oder** (mindestens) eine ist 1.

```
>> A = [1,2; 3,4]      % 2x2
>> A + 2               % Skalar (1x1): immer kompatibel
ans =
     3     4
     5     6
>> x = [2, 7]          % 1x2: kompatibel → x wird auf jede Zeile angewendet
>> A + x
ans =
     3     9
     5    11
```

→ Nützlich z. B. um von jeder Zeile einer Matrix einen Vektor zu subtrahieren.

Multiplikation von Vektoren und Matrizen

Ausdruck	Art	Bedeutung
$c * A$	Skalarmultiplikation	alle Elemente $\times c$
$A * B$	Matrizenmultiplikation	aus der linearen Algebra
$A .* B$	elementweise Multiplikation	$A(i,j) \times B(i,j)$
$A .^ n$	elementweise Potenz	$A(i,j)^n$
$A ./ B$	elementweise Division	$A(i,j) / B(i,j)$

Matrizenmultiplikation

- $A (n \times k) \times B (k \times m) = C (n \times m)$
- Element $C(p,q)$ = Skalarprodukt des p-ten Zeilenvektors von A mit dem q-ten Spaltenvektor von B

$$C(p, q) = \sum_{i=1}^k A(p, i) \cdot B(i, q)$$

- MATLAB führt die Matrizenmultiplikation automatisch durch – keine Schleifen nötig

- **Achtung:** Im Allgemeinen gilt $A \cdot B \neq B \cdot A$ (nicht kommutativ)

Matrizenmultiplikation – Beispiel

```
>> A = [1,2; 3,4; 5,6]           % 3x2
>> B = [10,11,12,13; 20,21,22,23] % 2x4
>> C = A * B                       % 3x4
C =
    50    53    56    59
   110   117   124   131
   170   181   192   203
```

$$C(1,1) = 1 \cdot 10 + 2 \cdot 20 = 50, C(2,3) = 3 \cdot 12 + 4 \cdot 22 = 124$$

Elementweise Multiplikation

```
>> A = [1,2; 3,4]
>> B = [10,11; 20,21]
>> C = A .* B           % C(i,j) = A(i,j) * B(i,j)
C =
    10    22
    60    84
>> D = 5 .* A           % = 5 * A (Skalarmultiplikation)
D =
     5    10
    15    20
```

Elementweise Operatoren: `.*` `./` `.^`

Operatoren – Vergleich MATLAB vs. Python/NumPy

Operation	MATLAB	Python / NumPy
Matrizenmultiplikation	$A * B$	$A @ B$
Elementweise Multiplikation	$A .* B$	$A * B$
Elementweise Potenz	$A .^ n$	$A ** n$
Elementweise Division	$A ./ B$	A / B

Colon-Operator – Sequenzen erzeugen

`start:step:end` % von start bis end mit Schrittweite step
`start:end` % Schrittweite = 1 (Default)

```
x = 1:5           % [1, 2, 3, 4, 5]
x = 1:2:10        % [1, 3, 5, 7, 9]
x = 10:-1:1       % [10, 9, ..., 1]
x = 0:pi/4:pi     % [0, 0.785, 1.571, 2.356, 3.142]
```

Äquidistante Werte mit bekannter Anzahl: `linspace`

```
x = linspace(0, pi, 100) % 100 Werte von 0 bis  $\pi$ 
```

→ In Python: `np.linspace(0, np.pi, 100)` – gleiche Semantik!

Übung: Arrays erzeugen [?]

Erzeugen Sie die folgenden Arrays – es gibt jeweils mehr als einen Weg. Schreiben Sie mindestens zwei Varianten auf.

a) Zeilenvektor: $(1 \ 2 \ 3 \ 4 \ 5)$

b) Spaltenvektor mit 4 Nullen: $\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$

c) 3×3 -Matrix, alle Einträge = 1

d) Zeilenvektor: $(10 \ 8 \ 6 \ 4 \ 2)$

Vektoren für die Funktion `plot` erzeugen

```
plot(x, y)
```

- `x` und `y` sind Vektoren mit je `n` Elementen
- `plot` zeichnet eine Kurve durch die Punkte $(x(k), y(k))$

Vorgehen: 1. Vektor `x` mit äquidistanten Elementen erzeugen (Colon-Operator oder `linspace`)

2. Funktionswerte `y` elementweise berechnen

```
x = 2.0:0.02:4.7;    % Vektor mit Schrittweite 0.02
y = sin(x) ./ x;     % elementweise berechnen
plot(x, y)
```

Vektoren erzeugen – Aufgaben

Erzeugen Sie einen Vektor x mit äquidistanten Elementen von 2.0 bis 4.7, Abstand 0.02. Wie viele Elemente besitzt der Vektor?

Berechnen Sie für diesen Vektor folgende Funktionswerte (elementweise) und zeichnen Sie die Funktion mit `plot(x, y)`:

- $f(x) = \sin(x)/x$
- $f(x) = e^{-x} \cdot \cos(x)$

```
x = 2.0:0.02:4.7;
length(x)           % Anzahl der Elemente
```

Übung: Code annotieren

Ein Kollege hat folgenden MATLAB-Code zur Auswertung einer Kraftmessung geschrieben.

Schreiben Sie zu jeder Zeile einen deutschen Kommentar – was berechnet sie, was bedeutet die Variable?

```
x      = linspace(0, 1.5, 6);           % ?
F      = [0, 420, 850, 1240, 830, 0];   % ?

F_max  = max(F);                        % ?
x_max  = x(F == F_max);                 % ?
F_norm = F / F_max;                    % ?
x_krit = x(F > 0.8 * F_max);           % ?
```

Vergleichen Sie Ihre Annotationen mit Ihrer Nachbarperson. Wo gibt es Unterschiede?

Kontext: Biegebalken der Länge 1,5 m, Kraft F in Newton gemessen an 6 Punkten.

Häufige Fehler

Fehler	Problem	Lösung
<code>A * B</code> statt <code>A .* B</code>	Matrizenmultiplikation statt elementweise	Punkt vor Operator
<code>A(0, 1)</code>	Indizes beginnen bei 1, nicht 0	<code>A(1, 1)</code>
<code>A(3,2)</code> lesen, obwohl nicht existent	Index außerhalb der Matrix	<code>size(A)</code> prüfen
<code>A(3,2) = 5</code> schreiben	Matrix wächst still – neue Elemente = 0	Bewusst einsetzen oder vermeiden
<code>x = [1,2,3]</code> statt <code>x = [1;2;3]</code>	Zeilen- statt Spaltenvektor	<code>;</code> für Spaltenvektor