

FORMELSAMMLUNG LINEARE ALGEBRA

Basierend auf Übungen FH Vorarlberg und dem LA-Kurs von Klaus Rheinberger
[pages.labs.fhv.at/~kr/lehre/la/]

1. VEKTOREN UND MATRIZEN

• Vektoroperationen:

- Addition: $\begin{bmatrix} a_1 \\ a_2 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} a_1+b_1 \\ a_2+b_2 \end{bmatrix}$
- Skalarprodukt: $a \cdot b = a_1b_1 + a_2b_2 + \dots + a_nb_n$
- Kreuzprod. (\mathbb{R}^3): $a \times b = [a_2b_3 - a_3b_2, a_3b_1 - a_1b_3, a_1b_2 - a_2b_1]$

• Matrizen:

- Multiplikation: $(AB)_{ij} = \sum A_{ik} \cdot B_{kj}$
- Transponierte: $(A^T)_{ij} = A_{ji}$
- Spur: $\text{tr}(A) = \sum A_{ii}$ (Diagonalsumme)

2. LINEARE GLEICHUNGSSYSTEME (LGS)

• Lösbarkeit:

$\text{Rang}(A) = \text{Rang}(A b) = n$	Eindeutige Lösung
$\text{Rang}(A) = \text{Rang}(A b) < n$	Unendlich viele Lösungen
$\text{Rang}(A) < \text{Rang}(A b)$	Keine Lösung

• Gauß-Elimination:

- Ziel: Zeilenstufenform (REF)
- Pivot-Elemente: Erste Nicht-Null-Elemente pro Zeile

3. DETERMINANTE & INVERSE

• 2x2-Matrix:

$$\det(A) = ad - bc \quad A^{-1} = \frac{1}{\det(A)} \cdot \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

• 3x3-Matrix (Sarrus):

$$\det(A) = aei + bfg + cdh - ceg - bdi - afh$$

4. EIGENWERTE & EIGENVEKTOREN

- Char. Gleichung: $\det(A - \lambda I) = 0$
- Eigenvektoren: $(A - \lambda I)v = 0$
- Diagonalisierung: $A = PDP^{-1}$ (D = Diagonalmatrix)

5. ORTHOGONALITÄT & PROJEKTION

- Projektion von b auf a :
 $\text{proj}_a b = (a \cdot b / a \cdot a) \cdot a$
- Orthogonale Matrix:
 $Q^T Q = I$ (Spalten orthonormal)

6. LINEARE REGRESSION

- Methode der kleinsten Quadrate:
 $x = (A^T A)^{-1} A^T b$
- Exponentieller Fit:
 $y = ce^{at} \rightarrow \ln(y) = \ln(c) + at$

7. PYTHON-BEFEHLE

<code>np.linalg.solve(A,b)</code>	Löst $Ax = b$
<code>np.linalg.inv(A)</code>	Inverse von A
<code>np.linalg.eig(A)</code>	Eigenwerte/-vektoren
<code>np.polyfit(x,y,deg)</code>	Polynomfit (Grad deg)

8. ANWENDUNGEN (AUS ÜBUNGEN)

- Ill-Conditioned Systems:
Kleine Änderungen in $b \rightarrow$ große Änderungen in x
- Adjazenzmatrizen:
 $C_{ij} = 1$ (wenn Kante $i \rightarrow j$ existiert)
- Leontief-Modell:
 $p = (I - A)^{-1} d$ (Produktionsplanung)

=====

NUMPY BEFEHLE FÜR LINEARE ALGEBRA

=====

1. MATRIXOPERATIONEN

<code>np.array([[1,2],[3,4]])</code>	# Matrix erstellen
<code>A.T</code>	# Transponierte
<code>np.linalg.inv(A)</code>	# Inverse (für quadratische Matrizen)
<code>np.linalg.det(A)</code>	# Determinante
<code>np.trace(A)</code>	# Spur (Summe der Diagonalelemente)
<code>np.eye(n)</code>	# Einheitsmatrix (n×n)
<code>np.diag([a,b,c])</code>	# Diagonalmatrix erstellen

2. LINEARE GLEICHUNGSSYSTEME

<code>np.linalg.solve(A, b)</code>	# Löse $Ax = b$ (für quadratische A)
<code>np.linalg.lstsq(A, b, rcond=None)</code>	# Least-Squares-Lösung (für nicht-quadratische A)
<code>np.linalg.matrix_rank(A)</code>	# Rang der Matrix
<code>np.linalg.pinv(A)</code>	# Pseudoinverse (Moore-Penrose-Inverse)

3. EIGENWERTE & ZERLEGUNGEN

<code>np.linalg.eig(A)</code>	# Eigenwerte und -vektoren
<code>np.linalg.eigvals(A)</code>	# Nur Eigenwerte
<code>np.linalg.qr(A)</code>	# QR-Zerlegung
<code>np.linalg.svd(A)</code>	# Singulärwertzerlegung (SVD)

4. VEKTOROPERATIONEN

<code>np.dot(a, b)</code>	# Skalarprodukt ($a \cdot b$)
<code>np.cross(a, b)</code>	# Kreuzprodukt (nur \mathbb{R}^3)
<code>np.linalg.norm(a)</code>	# Norm (Länge) des Vektors
<code>np.angle(v)</code>	# Winkel komplexer Vektoren
<code>np.outer(a, b)</code>	# Äußeres Produkt ($a \otimes b$)

5. SPEZIELLE MATRIZEN

<code>np.zeros((m,n))</code>	# Nullmatrix
<code>np.ones((m,n))</code>	# Einsen-Matrix
<code>np.random.rand(m,n)</code>	# Zufallsmatrix (Gleichverteilung)
<code>np.random.randn(m,n)</code>	# Zufallsmatrix (Normalverteilung)
<code>np.triu(A)</code>	# Oberes Dreieck
<code>np.tril(A)</code>	# Unteres Dreieck

6. PRAKTISCHE ANWENDUNGEN (AUS ÜBUNGEN)

Adjazenzmatrix-Analyse (Übung 4.5)

```
C @ C          # Matrixpotenz ( $C^2$  für 1 Umstieg)
np.linalg.matrix_power(C, k)  # Allgemeine Matrixpotenz

# Dynamische Systeme (Übung 7.1)
np.linalg.matrix_power(M, 50) # Langzeitverhalten nach 50 Schritten

# Lineare Regression (Übung 8.1)
np.polyfit(t, np.log(y), 1)   # Exponentieller Fit ( $y = ce^{\alpha t}$ )
np.polyval(coeffs, t_new)     # Auswertung des Polynoms
```

=====