Matrizen

Theorie

Version vom 12. März 2025

Inhaltsverzeichnis

Einleitung

Matrizen spielen eine zentrale Rolle in der wissenschaftlichen Datenverarbeitung und bieten vielfältige Anwendungsmöglichkeiten. Sie ermöglichen die effiziente Darstellung und Manipulation grosser Datenmengen in tabellarischer Form.

Hauptfunktionen von Matrizen

- *Datenrepräsentation:* Matrizen können mehrdimensionale Daten kompakt darstellen, was besonders nützlich für die Analyse komplexer Datensätze ist.
- *Berechnungseffizienz:* Durch Matrixoperationen können umfangreiche Berechnungen schnell und effizient durchgeführt werden, was besonders in der Datenanalyse und im maschinellen Lernen von Vorteil ist.
- *Mustererkennung:* Matrizen erleichtern die Identifikation von Mustern und Trends in grossen Datensätzen, was für viele wissenschaftliche Anwendungen unerlässlich ist.

Anwendungsbeispiele:

- *Maschinelles Lernen:* Matrizen bilden das Fundament vieler Algorithmen, ermöglichen Dimensionsreduktion, Merkmalsextraktion und lineare Regression.
- *Bildverarbeitung:* In der digitalen Bildverarbeitung werden Matrizen für Operationen wie Faltung, Filterung und Transformation eingesetzt.
- *Signalverarbeitung:* Matrizen werden zur Filterung, Umwandlung und Analyse von Audio-, Video- und Sensordaten verwendet.
- *Optimierungsprobleme:* Komplexe Optimierungsprobleme in Technik, Wirtschaft und Logistik können mithilfe von Matrizen formuliert und gelöst werden.
- *Neuronale Netze:* Die Implementierung neuronaler Netze basiert auf Matrixoperationen zur Darstellung von Gewichten, Eingaben und Aktivierungen.

Quellen und Informationen

- Die obige Einleitung wurde mit Hilfe von `www.perplexity.ai` erzeugt.
- Die Inhalte von Kapitel 1, 2 und 4 orientieren sich im Aufbau am Anhang D des Lehrbuchs *Introduction to Algorithms* von Cormen et. al. (2009).
- Die Python-Codefragmente sollten mit Python 3.x lauffähig sein; sind aber nicht optimiert. Für eine effiziente Python-Bibliothek für Matrizen siehe `www.numpy.org`.

1 Matrizen und Vektoren

Der Matrixbegriff

Eine Matrix A ist eine rechteckige Zahlentabelle. Die Matrix

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix} = \begin{pmatrix} 1 & 5 & 7 \\ 3 & 2 & 4 \end{pmatrix} \tag{1}$$

ist eine 2×3-Matrix $A = (a_{ij})$, wobei a_{ij} für $i = 1, 2$ und $j = 1, 2, 3$ das Matrixelement in Zeile i und Spalte j bezeichnet.

Python-Code:

```
A = [[1, 5, 7], [3, 2, 4]]
```

Wir verwenden Grossbuchstaben für Matrizen und entsprechenden Kleinbuchstaben für die Benennung ihrer Elemente.

$\mathbb{R}^{m\times n}$ bezeichnet die Menge aller $m\times n$-Matrizen mit Elementen aus der Menge der reellen Zahlen $\mathbb{R}$.

Transponierte einer Matrix

Die Transponierte einer Matrix A ist die Matrix A^T, die aus A hervorgeht, indem man die i-te Zeile zur i-ten Spalte macht. Für die Matrix A aus Gleichung (?).

$$A^\mathsf{T} = \begin{pmatrix} 1 & 2 \\ 5 & 3 \\ 7 & 4 \end{pmatrix} \tag{2}$$

Python-Code:

```
def transpose(A):
    '''Gibt die Transponierte der Matrix A zurück'''
    m, n = len(A), len(A[0])
    return [[A[i][j] for i in range(m)] for j in range(n)]
```

Vektoren

Ein Vektor ist eine eindmensionale Liste von Zahlen. Der Vektor

$$x = \begin{pmatrix} 3 \\ 2 \\ 5 \end{pmatrix} \tag{3}$$

ist ein Vektor mit drei Komponenten. Gelegentlich nennen wir einen Vektor mit n Komponenten einen n-Vektor. Wir verwenden Kleinbuchstaben für Vektoren und bezeichnen das i-te Element eines Vektors mit x_i für $i = 1, 2, \ldots, n$. Wir betrachten den Spaltenvektor als Standardform eines Vektors, der auch als $n\times1$-Matrix aufgefasst werden kann.

Den entsprechende Zeilenvektor erhalten wir durch Transposition

$$x^\mathsf{T} = \begin{pmatrix} 2 & 3 & 5 \end{pmatrix} \tag{4}$$

Python-Code:

```
x = [[2], [3], [5]]  # (3 x 1)-Matrix
print(transpose(x))  # => [[2, 3, 5]] (1 x 3)-Matrix
```

Der i-te Vektor der Standardbasis e_i ist der Vektor, dessen i-tes Element den Wert 1 hat und in dem alle anderen Element gleich 0 sind. Normalerweise ist die Grösse eines Einheitsvektors aus dem Kontext erkennbar.

Nullmatrix

Eine *Nullmatrix* ist eine Matrix, dessen Einträge alle 0 sind. Eine solche Matrix wird auch mit 0 bezeichnet, da eine Mehrdeutigkeit meist aus dem Kontext ausgeschlossen werden kann.

$$0 = \begin{pmatrix} 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{pmatrix} \tag{5}$$

Python-Code:

```
def zeromatrix(m, n):
    '''Gibt eine (m x n)-Nullmatrix zurück'''
    return [[0 for j in range(n)] for i in range(m)]
```

Quadratische Matrizen

Eine $(n \times n)$-Matrix wird *quadratische Matrix* genannt. Quadratische Matrizen treten häufig auf. Einige davon sind von besonderem Interesse.

Diagonalmatrizen

Für eine *Diagonalmatrix* gilt $a_{ij} = 0$, wenn $i \neq j$. Da alle Elemente ausserhalb der Diagonalen null sind, ist eine Diagonalmatrix durch ihrer Diagonalelement definiert:

$$\operatorname{diag}(a_{11}, a_{22}, \dots, a_{nn}) = \begin{pmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{12} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_{nn} \end{pmatrix} \tag{6}$$

Python-Code:

```
def diagonalmatrix(L):
    '''Gibt Diagonalmatrix mit Elementen aus Liste L zurück'''
    n = len(L)
    return [[L[i] if i==j else 0 for j in range(n)] for i in range(n)]
```

Einheitsmatrix

Die *Einheitsmatrix* I_n ist die Diagonalmatrix mit Einsen auf der Diagonalen.

$$I_n = \operatorname{diag}(1, 1, \ldots, 1) = \begin{pmatrix} 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 1 \end{pmatrix} \tag{7}$$

Fehlt der Index n, so ist die Grösse von I aus dem Kontext zu ermitteln.

Die i-te Spalte einer Einheitsmatrix ist der i-te Vektor der Standardbasis.

Python-Code:

```
def identity(n):
    '''Gibt n x n-Einheitsmatrix zurück'''
    return [[1 if i==j else 0 for j in range(n)] for i in range(n)]
```

Obere Dreiecksmatrizen

Eine *obere Dreiecksmatrix* ist eine Matrix U, deren Einträge oberhalb der Diagonalen alle null sind. Formal: $u_{ij} = 0$ für $i > j$.

$$U = \begin{pmatrix} u_{11} & u_{12} & \ldots & u_{1n} \\ 0 & u_{22} & \ldots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & u_{nn} \end{pmatrix} \tag{8}$$

Eine obere Dreiecksmatrix heisst *normierte obere Dreiecksmatrix*, falls alle Elemente auf der Diagonalen gleich 1 sind.

Untere Dreiecksmatrizen

Eine untere Dreicksmatrix L ist eine quadratische Matrix, für die alle Elemente oberhalb der Diagonalen gleich null sind. Formal: $\ell_{ij} = 0$ für $i < j$.

$$U = \begin{pmatrix} \ell_{11} & 0 & \ldots & 0 \\ \ell_{12} & \ell_{22} & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \ell_{n1} & \ell_{n2} & \ldots & \ell_{nn} \end{pmatrix} \tag{9}$$

Eine untere Dreiecksmatrix heisst *normierte untere Dreiecksmatrix*, falls alle Elemente auf der Diagonalen gleich 1 sind.

Permutationsmatrizen

Eine *Permutationsmatrix* P ist eine quadratische Matrix, die genau eine 1 in jeder Zeile und jeder Spalte hat. Ein Beispiel:

$$P = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}. \tag{10}$$

Multipliziert man eine Permutationsmatrix von links [rechts] mit einem Vektor bzw. einer Matrix, so werden die entsprechende Zeilen [Spalten] des Vektors bzw. der Matrix vertauscht.

Die obige Permutation kann auch kürzer als 5-Tupel

$$\pi = (2, 4, 1, 5, 3) \tag{11}$$

dargestellt werden, die das i-te Element an die $\pi(i)$-te Stelle setzt.

Python-Code:

```
1 def permutationmatrix(pi):
2     '''Gibt die Permutationsmatrix zur Tupeldarstellung pi zurück'''
3     n = len(pi)
4     return [[1 if j==pi[i] else 0 for j in range(n)] for i in range(n)]
```

Symmetrische Matrizen

Eine Matrix A mit der Eigenschaft $A = A^\mathsf{T}$ wird *symmetrische Matrix* genannt. Die Matrix

$$\begin{pmatrix} 2 & 5 & 7 \\ 5 & 4 & 3 \\ 7 & 3 & 6 \end{pmatrix} \tag{12}$$

ist symmetrisch.

2 Elementare Matrizenoperationen

Die algebraische Struktur der Elemente

Die Elemente von Matrizen und Vektoren stammen in der Regel aus einem Zahlenkörper wie dem der Menge der reellen Zahlen oder dem Restklassenköper modulo einer Primzahl. In diesem Fall können die elementaren algebraischen Operationen ausgeführt werden und es gelten das Kommutativgesetz, das Assoziativgesetz und das Distributivgesetz.

Einige dieser Operationen und Gesetze lassen sich auf Matrizen erweitern.

Matrizenaddition

Sind $A = (a_{ij})$ und $B = (b_{ij})$ $m \times n$-Matrizen, dann ist ihre Summe die $m \times n$-Matrix $C = (c_{ij}) = A + B$, die durch

$$c_{ij} = a_{ij} + b_{ij} \quad \text{für } i = 1, 2, \ldots, m \text{ und } j = 1, 2, \ldots, n \tag{13}$$

definiert wird.

Python-Code:

```
def add(A, B):
    '''Gibt die Summe der Matrizen A und B zurück'''
    m, n = len(A), len(A[0])
    return [[A[i][j]+B[i][j] for j in range(n)] for i in range(m)]
```

Die Nullmatrix ist das neutrale Element der Matrizenaddition.

$$A + 0 = A = 0 + A \tag{14}$$

Multiplikation von Matrizen mit Skalaren

Ist α eine Zahl und $A = a_{ij}$ eine Matrix, dann ist $\alpha A = (\alpha a_{ij})$ das *skalare Vielfache* von A, das durch Multiplikation von jedem Element von A mit α gewonnen wird.

Als Spezialfall definieren wir das Negative einer Matrix $A = (a_{ij})$ durch $-A = (-1) \cdot A = (-a_{ij})$.

Python-Code:

```
def smul(alpha, A):
    '''Gibt das Produkt der Zahl alpha mit der Matrix A zurück'''
    m, n = len(A), len(A[0])
    return [[alpha*A[i][j] for j in range(n)] for i in range(m)]
```

Für jede Matrix A ist $-A$ das inverse Element der Addition:

$$A + (-A) = 0 = (-A) + A\,. \tag{15}$$

Matrizensubtraktion

Wir verwenden das Negative einer Matrix, um die Subtraktion von Matrizen zu definieren:

$$A - B = A + (-B) \tag{16}$$

Python-Code:

```
def sub(A, B):
    '''Gibt die Differenz der Matrizen A und B zurück'''
    m, n = len(A), len(A[0])
    return [[A[i][j]-B[i][j] for j in range(n)] for i in range(m)]
```

Matrizenmultiplikation

Sind $A \in \mathbb{R}^{m\times r}$ und $B \in \mathbb{R}^{r\times n}$ Matrizen, so dass die Spaltenzahl von A mit der Zeilenzahl von B übereinstimmt, dann ist ihr Matrizenprodukt $C = AB$ die $m\times n$-Matrix $C = (c_{ij})$ mit

$$c_{ij} = \sum_{k=1}^{r} a_{ik} b_{kj} \tag{17}$$

für $i = 1, 2, \ldots, m$ und $j = 1, 2, \ldots, n$.

Python-Code:

```
def mul(A, B):
   '''Gibt das Matrizenprodukt der Matrizen A und B zurück'''
    m, r, n = len(A), len(B), len(B[0])
    return [[sum(A[i][k]*B[k][j] for k in range(r))
             for j in range(n)] for i in range(m)]
```

Multiplikation mit Einheitsmatrizen

Ist A eine $m\times n$-Matrix, so gilt

$$I_m\, A = A I_n = A \tag{18}$$

Multiplikation mit Nullmatrizen

Ist A eine $m\times r$-Matrix und 0 eine $r\times n$-Nullmatrix, so gilt

$$A\, 0 = 0' \tag{19}$$

wobei $0'$ eine $m\times n$-Nullmatrix ist.

Assoziativität

Die Matrizenmultiplikation ist assoziativ: Für alle passenden Matrizen A, B und C gilt

$$(AB)C = A(BC), \tag{20}$$

wobei wir für die Auswertung $(AB)C$ [„von links nach rechts“] auch die kürzere klammerlose Darstellung ABC verwenden.

Distributivgesetze

Für alle Matrizen A, B und C mit den passenden Dimensionen gelten die Distributivgesetze:

$$\begin{aligned} A(B + C) &= AB + AC \\ (A + B)C &= AC + BC \end{aligned} \tag{21}$$

Nicht-Kommutativität

Die Multiplikation von Matrizen ist *nicht kommutativ*.

- Ist A eine 2×3-Matrix und B eine 3×4-Matrix, so ist AB eine 2×4-Matrix aber BA ist nicht definiert.
- Ist A eine $m \times n$-Matrix und B eine $n \times m$-Matrix, so lassen sich AB und BA zwar berechnen, haben aber für $m \neq n$ nicht dieselbe Grösse und können dann auch nicht gleich sein.
- Sind A und B $n \times n$-Matrizen, so sind auch die Produkte $n \times n$-Matrizen, die im Allgemeinen aber nicht gleich sind:

$$\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$

Produkte mit Matrizen und Vektoren

Wir definieren Produkte zwischen Matrizen und Vektoren und zwischen Vektoren und Vektoren so, als wären die Vektoren $n \times 1$-Matrizen bzw. $1 \times n$-Matrizen, wenn es sich um Zeilenvektoren handelt. Damit gilt:

- Ist A ein $m \times n$-Matrix und x ein n-Vektor, dann ist Ax ein m-Vektor.

$$\begin{pmatrix} a_{11} & \dots & a_{1n} \\ a_{21} & \dots & a_{2n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} a_{11}x_1 + \dots + a_{1n}x_n \\ a_{21}x_1 + \dots + a_{2n}x_n \\ \vdots \\ a_{m1}x_1 + \dots + a_{mn}x_n \end{pmatrix} \tag{22}$$

- Sind x und y jeweils n-Vektoren, dann ist das *innere Produkt*

$$x^\mathsf{T} y = \begin{pmatrix} x_1 & \dots & x_n \end{pmatrix} \cdot \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} = x_1 y_1 + \dots + x_n y_n \tag{23}$$

  eine Zahl (genau genommen eine 1×1-Matrix).

- Sind x und y jeweils n-Vektoren, dann ist das *äussere Produkt*

$$x y^\mathsf{T} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \cdot \begin{pmatrix} y_1 & \dots & y_n \end{pmatrix} = \begin{pmatrix} x_1 y_1 & \dots & x_1 y_n \\ \vdots & \ddots & \vdots \\ x_n y_1 & \dots & x_n y_n \end{pmatrix} \tag{24}$$

  eine $n \times n$-Matrix.

Euklidsche Norm eines Vektors

Die euklische Norm $||x||$ eines n-Vektors x wird definiert durch

$$||x|| = \sqrt{x_1^2 + x_2^2 + \ldots + x_n^2} = \sqrt{x^\mathsf{T} x}\,. \tag{25}$$

$||x||$ ist die Länge von x im n-dimensionalen euklidischen Raum.

3 Lösen von Gleichungssystemen

Gleichungssysteme in Matrizenform

Die kompakte Darstellung linearer Gleichungssysteme (LGS) ist ein wichtiger Grund dafür, dass die Mulitplikation von Matrizen meist wie oben (und nicht komponentenweise) definiert wird.

$$\begin{aligned} 2x_1 + 4x_2 - \;\; x_3 &= 7 \\ x_1 - 3x_2 + 5x_3 &= 10 \\ 3x_1 \qquad + 7x_3 &= 24 \end{aligned} \tag{26}$$

$$\underbrace{\begin{pmatrix} 2 & 4 & -1 \\ 1 & -3 & 5 \\ 3 & 0 & 7 \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}}_{x} = \underbrace{\begin{pmatrix} 7 \\ 10 \\ 24 \end{pmatrix}}_{b} \tag{27}$$

A ist die *Koeffizientenmatrix* des Gleichungssystems $Ax = b$.

Gleichungssysteme in unterer Dreiecksform

$$\begin{aligned} 4x_1 &= 8 && \Rightarrow && x_1 \overset{(a)}{=} 8/4 = 2 \\ 2x_1 + 3x_2 &= 7 && \Rightarrow && x_2 \overset{(b)}{=} (7 - 2\cdot 2)/3 = 1 \\ x_1 + 5x_2 + 2x_3 &= 9 && \Rightarrow && x_3 \overset{(c)}{=} (9 - 1\cdot 2 - 5\cdot 1)/2 = 1 \end{aligned}$$

Ist die Koeffizientenmatrix eine untere Dreiecksmatrix L, so löst man die Gleichung $Lx = b$ durch *Vorwärtseinsetzen*:

$$x_i = \left(b_i - \sum_{j=1}^{i-1} \ell_{ij} \right) / \ell_{ii} \quad \text{für } i = 1, \ldots, n \text{ und } \ell_{ii} \neq 0 \tag{28}$$

Python-Code

```
1  def forward_sub(L, b):
2      '''Gibt Lösung von Lx=b für untere Dreiecksmatrix L zurück'''
3      n = len(L)
4      x = [0 for i in range(n)]
5      for i in range(0, n):
6          x[i] = (b[i]-sum(L[i][j]*x[j] for j in range(i)))/L[i][i]
7      return x
```

Gleichungssysteme in oberer Dreiecksform

$$\begin{aligned} x_1 + 5x_2 + 2x_3 = 9 \quad &\Rightarrow \quad x_1 \overset{(c)}{=} (9 - 2 \cdot 2 - 5 \cdot 0)/1 = 5 \\ 2x_2 + 3x_3 = 6 \quad &\Rightarrow \quad x_2 \overset{(b)}{=} (6 - 3 \cdot 2)/2 = 0 \\ 4x_3 = 8 \quad &\Rightarrow \quad x_3 \overset{(a)}{=} 8/4 = 2 \end{aligned}$$

Ist die Koeffizientenmatrix eine obere Dreiecksmatrix U, so löst man die Gleichung $Ux = b$ durch *Rückwärtseinsetzen*:

$$x_i = \left(b_i - \sum_{j=i+1}^{n} u_{ij} \right) / u_{ii} \quad \text{für } i = n, \ldots, 1 \text{ und } u_{ii} \neq 0 \tag{29}$$

Python-Code

```
def back_sub(U, b):
    '''Gibt Lösung von Lx=b für obere Dreiecksmatrix U zurück'''
    n = len(U)
    x = [0 for i in range(n)]
    for i in range(n-1, -1, -1):
        x[i] = (b[i]-sum(U[i][j]*x[j] for j in range(i,n)))/U[i][i]
    return x
```

LU-Zerlegung

Die *LU-Zerlegung* einer $(n \times n)$-Matrix A ist ein Paar von $(n \times n)$-Matrizen L und U mit folgenden Eigenschaften:

(1) $A = LU$

(2) L ist eine untere Diagonalmatrix mit $\ell_{ii} = 1$ $(i = 1, \ldots, n)$

(3) U ist eine obere Diagonalmatrix

Eigenschaften der LU-Zerlegung

- Es gibt Matrizen A, für die keine LR-Zerlegung existiert.
- Gibt es eine LR-Zerlegung, so ist sie eindeutig.
- Mit einer LR-Zerlegung kann ein LGS effizient gelöst werden.
- Die Einsen auf der Diagonale in (2) sind der Grund für die Eindeutigkeit der Zerlegung. Wir hätten die Einsen stattdessen für (3) verlangen können aber das ist nicht üblich.

Beispiel (Zerlegung durch Koeffizientenvergleich)

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 4 & 3 \\ 1 & 1 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ \ell_{21} & 1 & 0 \\ \ell_{31} & \ell_{32} & 1 \end{pmatrix} \cdot \begin{pmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{pmatrix}$$

$$= \begin{pmatrix} \underset{1}{u_{11}} = 1 & \underset{1}{u_{12}} = 1 & \underset{1}{u_{13}} = 1 \\ \underset{1\ 1}{\ell_{21}u_{11}} = 1 & \underset{1\ 1}{\ell_{21}u_{12}} + \underset{3}{u_{22}} = 4 & \underset{1\ 1}{\ell_{21}u_{13}} + \underset{2}{u_{23}} = 3 \\ \underset{1\ 1}{\ell_{31}u_{11}} = 1 & \underset{1\ 1}{\ell_{31}u_{12}} + \underset{0\ 3}{\ell_{32}u_{22}} = 1 & \underset{1\ 1}{\ell_{31}u_{13}} + \underset{0\ 2}{\ell_{32}u_{23}} + \underset{1}{u_{33}} = 2 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 & 1 \\ 0 & 3 & 2 \\ 0 & 0 & 1 \end{pmatrix} \mathrel{\widehat{=}} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 3 & 2 \\ 1 & 0 & 1 \end{pmatrix}$$ speichersparende Darstellung

Wir werden bald effizientere Zerlegungsalgorithmen kennen lernen.

Bemerkung: Wäre der Eintrag $a_{11} = 0$ statt $a_{11} = 1$, so kann die LR-Zerlegung nicht auf die obige Weise bestimmt werden, obwohl die Matrix invertierbar ist.

Lösung eines LGS mit Hilfe der LU-Zerlegung

$$\begin{aligned} Ax &= b \\ LUx &= b \quad \text{substituiere } Ux = y \\ Ly &= b \quad \text{löse } Ly = b \text{ für } y \text{ mit Vorwärtseinsetzen} \\ Ux &= y \quad \text{löse } Ux = y \text{ für } x \text{ mit Rückwärtseinsetzen} \end{aligned}$$

Beispiel

$$\begin{pmatrix}1&1&1\\1&4&3\\1&1&2\end{pmatrix}\cdot\begin{pmatrix}x_1\\x_2\\x_3\end{pmatrix}=\begin{pmatrix}4\\7\\4\end{pmatrix}$$

$$\begin{pmatrix}1&0&0\\1&1&0\\1&0&1\end{pmatrix}\cdot\underbrace{\begin{pmatrix}1&1&1\\0&3&2\\0&0&1\end{pmatrix}\cdot\begin{pmatrix}x_1\\x_2\\x_3\end{pmatrix}}_{y}=\begin{pmatrix}4\\7\\4\end{pmatrix}$$ Zerlegung siehe oben

$$\begin{pmatrix}1&0&0\\1&1&0\\1&0&1\end{pmatrix}\cdot\begin{pmatrix}y_1\\y_2\\y_3\end{pmatrix}=\begin{pmatrix}4\\7\\4\end{pmatrix}\quad\Rightarrow\quad\begin{aligned}y_1&=4\\y_2&=3\\y_3&=0\end{aligned}$$ (Vorwärtseinsetzen)

$$\begin{pmatrix}1&1&1\\0&3&2\\0&0&1\end{pmatrix}\cdot\begin{pmatrix}x_1\\x_2\\x_3\end{pmatrix}=\begin{pmatrix}4\\3\\0\end{pmatrix}\quad\Rightarrow\quad\begin{aligned}x_1&=3\\x_2&=1\\x_3&=0\end{aligned}$$ (Rückwärtseinsetzen)

Die LU-Zerlegung berechnen

Der Kern der LU-Zerlegung ist das Gausssche Eliminationsverfahren, welche das Gleichungssystem systematisch durch Äquivalenzumformungen (Umformungen, welche die Lösungmenge nicht verändern) in eine Form bringt, aus der sich die Lösungen ablesen oder durch Vorwärtseinsetzen bestimmen lassen. Wenn wir die Umformungsschritte geschickt protokollieren, erhalten wir ohne grossen zusätzlichen Aufwand eine LR-Zerlegung.

Beispiel

Wir subtrahieren geeignete Vielfache der ersten Zeile von den folgenden Zeilen, damit alle Werte unterhalb von a_{11} (dem *Pivotelement*) gleich 0 werden. Dies wiederholen wir für die folgenden Diagonalelemente, bis wir eine obere Dreiecksmatrix erhalten.

$$\begin{array}{cccc}\boxed{2}&3&1&5\\6&13&5&19\\2&19&10&23\\4&10&11&31\end{array}\quad\begin{array}{l}\\\cdot 3,\ -\\\\\end{array}\quad\begin{array}{l}\\\\\cdot 1,\ -\\\end{array}\quad\begin{array}{l}\\\\\\\cdot 2,\ -\end{array}$$

$$\begin{array}{cccc}2&3&1&5\\0&\boxed{4}&2&4\\0&16&9&18\\0&4&9&21\end{array}\quad\begin{array}{l}\\\cdot 4,\ -\\\\\end{array}\quad\begin{array}{l}\\\\\cdot 1,\ -\\\end{array}$$

$$\begin{array}{cccc}2&3&1&5\\0&4&2&4\\0&0&\boxed{1}&2\\0&0&7&17\end{array}\quad\begin{array}{l}\\\\\cdot 7,\ -\end{array}$$

$$\begin{array}{cccc}2&3&1&5\\0&4&2&4\\0&0&1&2\\0&0&0&\boxed{3}\end{array}$$

Soweit haben wir nur das gausssche Eliminationsverfahren angewendet. Statt die Werte unter den Pivotelementen mit Nullen zu überschreiben, speichern wir dort die Elemente der unteren Dreiecksmatrix L. Dies sind die (blauen) Faktoren, mit denen wir die Pivotzeilen multiplizieren, bevor sie von den darunterliegenden Zeilen subtrahiert werden. Damit erhalten wir die LR-Zerlegung:

$$\begin{matrix} 2 & 3 & 1 & 5 \\ 3 & 4 & 2 & 4 \\ 1 & 4 & 1 & 2 \\ 2 & 1 & 7 & 3 \end{matrix}$$

LR-Zerlegung (kompakt)

oder ausgeschrieben:

$$\Rightarrow \begin{pmatrix} 2 & 3 & 1 & 5 \\ 6 & 13 & 5 & 19 \\ 2 & 19 & 10 & 23 \\ 4 & 10 & 11 & 31 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 \\ 1 & 4 & 1 & 0 \\ 2 & 1 & 7 & 1 \end{pmatrix} \begin{pmatrix} 2 & 3 & 1 & 5 \\ 0 & 4 & 2 & 4 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 3 \end{pmatrix}$$

4 Eigenschaften von Matrizen

In this section, we define some basic properties pertaining to Matrizen: inverses, linear dependence und independence, rank, und determinants. We also define the class of positive-definite Matrizen.

Inverse von Matrizen

Ist A eine $n \times n$-Matrix A und B eine $n \times n$-Matrix mit der Eigenschaft

$$AB = BA = I_n \,, \tag{30}$$

so wird B die *Inverse* der Matrix A genannt und mit A^{-1} bezeichnet.

Eine Matrix A, zu der es eine Inverse gibt, wird *invertierbar* oder *regulär* genannt. Eine Matrix A, die keine Inverse besitzt, wird *nicht invertierbar* oder *singulär* genannt.

Beispiele

(a) $A = \begin{pmatrix} 2 & 1 \\ 5 & 3 \end{pmatrix}$ hat die Inverse $A^{-1} = \begin{pmatrix} 3 & -1 \\ 5 & 2 \end{pmatrix}$, denn

$$AA^{-1} = \begin{pmatrix} 2 & 1 \\ 5 & 3 \end{pmatrix} \begin{pmatrix} 3 & -1 \\ 5 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \text{ und } A^{-1}A = \begin{pmatrix} 3 & -1 \\ 5 & 2 \end{pmatrix} \begin{pmatrix} 2 & 1 \\ 5 & 3 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

(b) $A = \begin{pmatrix} 2 & 3 \\ 0 & 0 \end{pmatrix}$ ist nicht invertierbar, denn

$$\begin{pmatrix} 2 & 3 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} r & s \\ t & u \end{pmatrix} = \begin{pmatrix} 2r + 3t & 2s + 3u \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

ist für keine Werte von r, s, t, u erfüllbar.

Rechenregeln für invertierbare Matrizen

(a) Wenn die Inverse einer Matrix A existiert, so ist sie eindeutig.

(b) Sind A und B invertierbar, dann ist auch AB invertierbar und es gilt:

$$(AB)^{-1} = B^{-1}A^{-1}\,. \tag{31}$$

(c) Ist A invertierbar, dann ist auch A^T invertierbar und es gilt:

$$\left(A^\mathsf{T}\right)^{-1} = \left(A^{-1}\right)^\mathsf{T} \tag{32}$$

Lineare Unabhängigkeit

Die n Vektoren x_1, x_2, …, x_n sind *linear abhängig*, wenn es Zahlen c_1, c_2, …, c_n gibt, die nicht alle null sind und die Gleichung

$$c_1x_1 + c_2x_2 + \cdots + c_nx_n = 0 \tag{33}$$

erfüllen. Andernfalls sind die Vektoren *linear unabhängig*.

Beispiele

(a) Die Zeilenvektoren $x_1 = \begin{pmatrix}3 & 2 & 4\end{pmatrix}$, $x_2 = \begin{pmatrix}1 & 1 & 5\end{pmatrix}$ und $x_3 = \begin{pmatrix}7 & 4 & 2\end{pmatrix}$ sind wegen $3x_1 - 2x_2 - x_3 = 0$ linear abhängig.

(b) Die Spaltenvektoren einer 3×3 Einheitsmatrix sind linear unabhängig, da sich

$$c_1 \begin{pmatrix}1\\0\\0\end{pmatrix} + c_2 \begin{pmatrix}0\\1\\0\end{pmatrix} + c_3 \begin{pmatrix}0\\0\\1\end{pmatrix} = \begin{pmatrix}0\\0\\0\end{pmatrix}$$

offensichtlich nur für $c_1 = c_2 = c_3 = 0$ erfüllen lässt.