

Vorlesungsnotizen zu Numerik für Informatiker

Nicolas Gres

2025-06-06

Inhaltsverzeichnis

Einführung	3
1 Arithmetik	4
1.1 Gleitkommazahlen	4
1.2 Auslöschung	4
1.3 Vektor- und Matrixnormen	5
2 Direkte Lösungsverfahren für lineare Gleichungen	7
2.1 Vorwärts-Substitution	7
2.2 Rückwärts-Substitution	7
2.3 LR-Zerlegung	8
2.4 Choelsky-Zerlegung	10
2.4.1 Berechnung	11
2.5 QR-Zerlegung	12
Appendix	14
Mathematische Grundlagen	14
Reihen und Summen	14
Lineare Algebra	14
Referenzen	15

Einführung

Es handelt sich hierbei um meine Vorlesungsnotizen, basierend auf den Übungsaufzeichnungen, dem offiziellen Skript (Wieners 2025), sowie Passagen aus Bartels (2016).

Die Notizen sind nicht vollständig und dienen lediglich als Ergänzung zu den Vorlesungsunterlagen.

1 Arithmetik

1.1 Gleitkommazahlen

Wir betrachten für eine gegebene Basis $B \geq 2$, einen minimalen Exponent E^- und Längen M und E die endliche Menge der normalisierten Gleitpunktzahlen FL.

$$\text{FL} := \{\pm B^e \underbrace{\sum_{l=1}^M a_l B^{-l}}_{=m} \mid e = E^- + \sum_{k=0}^{E-1} c_k B^k, a_l, c_k \in \{0, \dots, B-1\}, a_1 \neq 0\} \cup \{0\}$$

1.2 Auslöschung

```
N = 2**10

def exp(x):
    """
    Compute the exponential function using Taylor series expansion.
    """
    return np.sum([x**n / math.factorial(n) for n in range(N)], axis=0)

x = 10

z_bad = exp(-x)
z_good = 1 / exp(x)

r = np.exp(-x) # reference

np.abs(z_bad - r) / r, np.abs(z_good - r) / r
```

```
(np.float64(6.529424994681785e-09), np.float64(1.4925713791816933e-16))
```

Quadratische Gleichung

Anstatt $x_2 = p - \sqrt{p^2 - q}$, verwenden wir

$$x_2 = p - \sqrt{p^2 - q} \cdot \frac{p + \sqrt{p^2 + q}}{p + \sqrt{p^2 + q}} = \frac{q}{p + \sqrt{p^2 - q}} = \frac{q}{x_1}$$

(Satz von Vieta) um die Auslöschung zwischen p und $\sqrt{p^2 - q}$ zu vermeiden.

```
p = 1e10
q = 1e2

print(np.roots([1, -2*p, q])) # reference

x1 = p + math.sqrt(p**2 - q)

x2_bad = p - math.sqrt(p**2 - q)
x2_good = q / x1

x2_bad, x2_good
```

[2.e+10 5.e-09]

(0.0, 5e-09)

1.3 Vektor- und Matrixnormen

Wir verwenden für $x \in \mathbb{R}^N$ und $A \in \mathbb{R}^{M \times N}$

$ x _1 = \sum_{n=1}^N x_n $	1-Norm
$ x _2 = \sqrt{x^T x} = \left(\sum_{n=1}^N x_n ^2 \right)^{\frac{1}{2}}$	Euklidische Norm
$ x _\infty = \max_{n=1, \dots, N} x_n $	Supremumsnorm
$\ A\ _1 = \max_{n=1, \dots, N} \sum_{m=1}^M A[m, n] $	Spaltensummennorm,
$\ A\ _2 = \sqrt{\rho(A^T A)}$	Spektralnrm,
$\ A\ _\infty = \max_{m=1, \dots, M} \sum_{n=1}^N A[m, n] $	Zeilensummennorm,
$\ A\ _F = \left(\sum_{m=1}^M \sum_{n=1}^N A[m, n] ^2 \right)^{\frac{1}{2}}$	Frobeniusnorm.

Dabei ist

$$\begin{aligned}\rho(A) &= \max\{|\lambda| : \lambda \in \sigma(A)\} \text{ Spektralradius,} \\ \sigma(A) &= \{\lambda \in \mathbb{C} : \det(A - \lambda I_N) = 0\} \text{ Spektrum.}\end{aligned}$$

Es gilt immer $|Ax|_p \leq \|A\|_p |x|_p$ für alle $x \in \mathbb{R}^N$ und wegen $\|A\|_2 \leq \|A\|_F$ auch

$$|Ax|_2 \leq \|A\|_2 |x|_2 \leq \|A\|_F |x|_2$$

2 Direkte Lösungsverfahren für lineare Gleichungen

2.1 Vorwärts-Substitution

Die *Vorwärts-Substitution* löst $L \cdot \mathbf{y} = \mathbf{b}$, indem wir über die Zeilen iterieren und dabei die Lösungen der vorherigen \mathbf{x}_j für die Berechnung des aktuellen \mathbf{x}_i verwenden ($\mathbf{x}_1 = \mathbf{b}_1$). Die Laufzeit liegt somit in $O(n^2)$.

```
def forward_sub(lower, rhs):
    n = lower.shape[0]
    solution = np.zeros(n)
    for i in range(n):
        solution[i] = rhs[i]
        for j in range(i):
            solution[i] -= lower[i, j] * solution[j]
        solution[i] = solution[i] / lower[i, i]
    return solution
```

```
forward_sub(np.array([
    [1, 0, 0],
    [2, 1, 0],
    [3, 4, 1]]
), np.array([1, 2, 3]))
```

```
array([1., 0., 0.])
```

2.2 Rückwärts-Substitution

Die *Rückwärts-Substitution* löst $R \cdot \mathbf{x} = \mathbf{y}$, indem wir von der letzten Zeile aus das verfahren der Vorwärts-Substitution anwenden. Die Laufzeit liegt somit ebenfalls in $O(n^2)$.

```
def backward_sub(upper, rhs):
    n = upper.shape[0]
    solution = np.zeros(n)
    for i in range(n - 1, -1, -1):
        tmp = rhs[i]
        for j in range(i + 1, n):
            tmp -= upper[i, j] * solution[j]
        solution[i] = tmp / upper[i, i]
    return solution
```

```
backward_sub(np.array([
    [2, 2, 3],
    [0, 1, 4],
    [0, 0, 1]]
), np.array([1, 0, 0]))
```

```
array([0.5, 0. , 0. ])
```

2.3 LR-Zerlegung

(en. *LU-Decomposition*)

<https://www.youtube.com/watch?v=BFYFkn-eQQk>



Warnung

Die 1-en auf der Diagonalen der L -Matrix bleiben beim Zeilentauschen **unverändert**

Die LR -Zerlegung lässt sich mittels des Gauß-Algorithmus bestimmen, indem wir A auf eine untere Dreiecksmatrix R gaußen und uns die Operationen in L “merken”. Sie ist eindeutig und benötigt $O(n^3)$ Operationen.

Die Berechnung ist **nicht** stabil.

Hinreichende Bedingungen für die Existenz einer LR -Zerlegung für eine **quadratische** Matrix A :

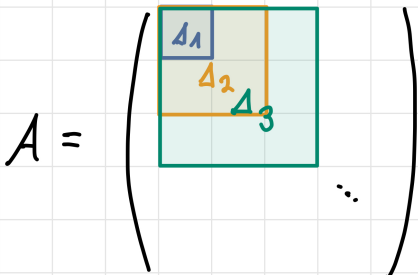
1. strikt diagonal-dominant, daher das Diagonalelement ist größer als die Summe aller anderen Elemente in der Zeile, bzw.

$$|A[n, n]| > \sum_{\substack{k=1 \\ k \neq n}}^N |A[n, k]| \quad \text{für } n = 1, \dots, N$$

2. positiv definit, daher **alle Eigenwerte** > 0 , bzw.

$$x^\top A x > 0 \quad \text{für alle } x \in \mathbb{R}^N, x \neq 0.$$

i Hinweis



$$A = \begin{pmatrix} \Delta_1 & & \\ & \Delta_2 & \\ & & \Delta_3 & \\ & & & \ddots \end{pmatrix} \in \mathbb{R}^{n \times n}$$

A pos. definit \Leftrightarrow
 $\det(\Delta_i) > 0 \forall i = 1, \dots, n$

Abbildung 2.1: Hauptminorenkriterium für positiv definite Matrizen

Falls diese Bedingungen nicht gegeben sind, können wir mittels **Zeilenvertauschung** (Permutationsmatrix P) eine LR-zerlegbare Matrix PA in $O(n^3)$ erzeugen.

```
def lu_decomposition(matrix):
    n = matrix.shape[0]
    lower = np.zeros(shape=matrix.shape)
    upper = np.zeros(shape=matrix.shape)
    for j in range(n):
        lower[j][j] = 1.0
        for i in range(j + 1):
            first_sum = sum(upper[k][j] * lower[i][k] for k in range(i))
            upper[i][j] = matrix[i][j] - first_sum
        for i in range(j, n):
            second_sum = sum(upper[k][j] * lower[i][k] for k in range(j))
            lower[i][j] = (matrix[i][j] - second_sum) / upper[j][j]
    return lower, upper

def solve_with_lu(matrix, rhs):
```

```

lower, upper = lu_decomposition(matrix)
y = forward_sub(lower, rhs)
return backward_sub(upper, y)

```

```

matrix = np.array([[2.0, 1.0],
[1.0, 4.0]])
rhs = np.array([1.0, 2.0])
solution = solve_with_lu(matrix, rhs)
print("solution", solution)
test = rhs - np.dot(matrix, solution)
print("test ", test)

```

```

solution [0.5 0. ]
test [0. 1.5]

```

2.4 Cholesky-Zerlegung

💡 (2.7) Satz

Sei $A \in \mathbb{R}^{N \times N}$ **symmetrisch** und **positiv definit**. Dann existiert genau eine Cholesky-Zerlegung $A = LL^T$ mit einer regulären unteren Dreiecksmatrix L .

Es handelt sich somit um eine Spezialisierung der LR-Zerlegung für symmetrisch, positiv definite Matrizen.

$$A = \begin{pmatrix} a_{11} & a_{21} & a_{31} \\ a_{21} & a_{22} & a_{32} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \quad (2.1)$$

$$= \begin{pmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{pmatrix} \begin{pmatrix} l_{11} & l_{21} & l_{31} \\ 0 & l_{22} & l_{32} \\ 0 & 0 & l_{33} \end{pmatrix} \equiv LL^T \quad (2.2)$$

$$= \begin{pmatrix} l_{11}^2 & l_{21}l_{11} & l_{31}l_{11} \\ l_{21}l_{11} & l_{21}^2 + l_{22}^2 & l_{31}l_{21} + l_{32}l_{22} \\ l_{31}l_{11} & l_{31}l_{21} + l_{32}l_{22} & l_{31}^2 + l_{32}^2 + l_{33}^2 \end{pmatrix} \quad (2.3)$$

2.4.1 Berechnung

Diagonalelemente:

$$l_{kk} = \sqrt{a_{kk} - \sum_{j=1}^{k-1} l_{kj}^2}$$

Rest:

$$l_{ik} = \frac{1}{l_{kk}} \left(a_{ik} - \sum_{j=1}^{k-1} l_{ij} l_{kj} \right)$$

```
def cholesky_decomposition(A):
    n = matrix.shape[0]
    lower = np.zeros(matrix.shape)
    lower[0, 0] = np.sqrt(matrix[0, 0])
    for n in range(1, n):
        y = forward_sub(lower[:n, :n], matrix[n, :n]) # linalg.solve_triangular(lower[:n, :n], matrix[n, :n], lowertriangular=True)
        lower[n, :n] = y
        lower[n, n] = np.sqrt(matrix[n, n] - np.dot(y, y))
    return lower

def solve_with_cholesky(matrix, rhs):
    lower = cholesky_decomposition(matrix)
    y = forward_sub(lower, rhs)
    return backward_sub(lower.transpose(), y)
```

```
matrix = np.array([[2.0, 1.0],
[1.0, 4.0]])
rhs = np.array([1.0, 2.0])
rhs = np.array([1.0, 2.0])
solution = solve_with_cholesky(matrix, rhs)
print("solution", solution)
test = rhs - np.dot(matrix, solution)
print("test ", test)
```

```
solution [0.70710678 0.          ]
test [-0.41421356  1.29289322]
```

- Die Cholesky-Zerlegung ist stabil: Es gilt $\kappa_2(L)^2 = \kappa(A)$

- Die Berechnung der Cholesky-Zerlegung benötigt nur halbsoviele Operationen wie die Berechnung einer LR-Zerlegung.
- Matrizen mit einer geeigneten Hüllenstruktur (viele Nullelemente wie bei der Bandmatrix) können effizienter gelöst werden (Bandmatrix in $O(NM^2)$)

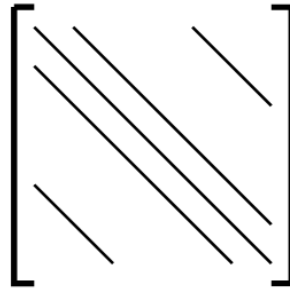


Abbildung 2.2: Schematische Darstellung einer Bandmatrix

2.5 QR-Zerlegung

💡 (2.14) Satz (QR-Zerlegung)

Zu $A \in \mathbb{R}^{M \times N}$ existiert eine QR-Zerlegung $A = QR$ in eine orthogonale Matrix $Q \in \mathbb{R}^{M \times M}$ mit $Q^\top Q = I_M$ und eine obere Dreiecksmatrix $R \in \mathbb{R}^{M \times N}$ mit $R[m, n] = 0$ für $m > n$.

- Das LGS $Ax = b$ kann durch die Berechnung $y = Q^\top b$ und darauf mit Rücksubstitution $Rx = y$ gelöst werden.
- Asymptotischer Aufwand in $O(N^3)$

Rotationen und Drehungen sind orthogonale Matrizen $Q \in \mathbb{R}^{N \times N}$ mit

- $QQ^\top = I_N$, $Q^\top Q = I_N$, so dass $Q^{-1} = Q^\top$,
- $|Qv|_2 = |v|_2$ und $(Qv)^\top(Qw) = v^\top w$ Längen und Winkel erhaltend,
- $\kappa_2(Q) = 1$.

$$A = \begin{pmatrix} | & & \\ v_1 & \dots & \\ | & & \end{pmatrix} \in \mathbb{R}^{M \times N}$$

Wähle $\sigma_i := \begin{cases} -\|v_i\|_2 & \text{falls } v_i[1] > 0 \\ \|v_i\|_2 & \text{sonst} \end{cases}$

$$w_i := v_i - \sigma_i \cdot e_1$$

$$\tilde{Q}_i := I_{(M+1-i)} - \frac{2}{\|w_i\|_2} w_i \cdot w_i^T$$

$$Q_i := \begin{pmatrix} 1 & & 0 \\ & \ddots & \\ 0 & & \tilde{Q}_i \end{pmatrix} \in \mathbb{R}^{M \times M}$$

$$X \cdot X^T = \begin{pmatrix} | & & | \\ X[1]X & \dots & X[N]X \\ | & & | \end{pmatrix}$$

$(Q_1 = \tilde{Q}_1, \text{ da bereits } \tilde{Q}_1 \in \mathbb{R}^{M \times M})$

$$Q_i \cdot \dots \cdot Q_1 \cdot A = \begin{pmatrix} * & \dots & * & \dots \\ | & & | & \dots \\ 0 & \dots & v_{i+1} & \dots \\ | & & | & \dots \end{pmatrix} \begin{matrix} i \text{ Zeilen} \\ \\ i+1\text{-te} \\ \text{Spalte} \end{matrix}$$

Wir wenden die vorherigen Transformationen auf A an und wählen den $(i+1)$ -ten Spaltenvektor ohne die ersten i Zeilen aus.

$$Q = Q_N \cdot \dots \cdot Q_1$$

Wiederhole bis zur letzten Spalte.
(die resultierende Matrix ist R)

$$R = Q \cdot A$$

Abbildung 2.3: QR-Zerlegung berechnen

Appendix

Mathematische Grundlagen

Reihen und Summen

Analysis

Für alle reellen $q \neq 1$ und für alle $n \in \mathbb{N}_0$ ist:

$$\sum_{k=0}^n q^k = \frac{1 - q^{n+1}}{1 - q}$$

Der Grenzwert ist dementsprechend:

$$\sum_{k=0}^{\infty} q^k = \frac{1}{1 - q}$$

Lineare Algebra

2×2 -Matrix invertieren

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \quad \text{then} \quad A^{-1} = \frac{1}{ad - bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

Referenzen

- Bartels, Sören. 2016. *Numerik 3x9: Drei Themengebiete in jeweils neun kurzen Kapiteln*. 1. Aufl. 2016. Springer-Lehrbuch. Berlin Heidelberg: Springer Spektrum. <https://doi.org/10.1007/978-3-662-48203-2>.
- Wieners, Christian. 2025. „Einführung in Die Numerische Mathematik“.