# Wissenschaftliches Rechnen

Skript SoS 2024

Dr. Dirk Frettlöh
Technische Fakultät
Universität Bielefeld

Kapitel 1-9 sind sehr nah am Skript von Mario Botsch

May 14, 2024

# Contents

# Vorab:

Früher wurde diese Vorlesung von Prof. Mario Botsch gehalten. Ich danke ihm für das viele Material. Kapitel 1-9 sind fast komplett aus seinem hervorragenden Skript übernommen. Perfekte Dinge kann und soll man nicht verbessern.

Dieser Text enthält bestimmt noch etliche größere und kleinere Fehler. Für jeden Hinweis auf Letztere bin ich dankbar.

# 1 Einführung

9. April

Wissenschaftliches Rechnen ist nicht scharf definiert, aber es liegt an der Schnittstelle zwischen Natur- und Ingenieurswissenschaften, Informatik und Mathe. Es benutzt Methoden der Informatik und angewandten Mathematik. Allgemein werden oft Simulation und Modellierung darunter verstanden. Innerhalb der Mathematik gibt es den Zweig der *Numerik*, der sich dem näherungsweisen Berechnen von allem Möglichen widmet. Ziel: möglichst genau, effizient und stabil.

Ausgehend von einem realen Problem ist der Ablauf oft so:

- Das Problem in ein mathematisches Modell übersetzen.
- Das Modell diskretisieren (d.h. endlich machen); dies liefert ganz oft ein lineares Gleichungssytem (LGS).
- Ein effizientes Computerprogramm schreiben bzw. nutzen, um das effizient zu lösen.
- Den Programmlauf beschleunigen durch kluges Programmieren und Compilieren, Parallelisieren usw.

Oft läuft es also auf **effizientes Lösen von LGS** hinaus. Das wird das erste große Thema. Zwei Themenfelder, wo dies Anwendung findet, sind **Interpolation und Approximation von Funktionen** sowie (indirekt) die **diskrete Fouriertransformation** (DFT). Das sind die beiden anderen großen Themen. Da ich selbst leider nur Experte für die Theorie bin, aber nicht für schnellen Code, wird der vierte Punkt von oben hier nicht behandelt. Dafür werden die Videos von Prof. Mario Botsch (der für alles hier Experte ist) zu diesem Punkt zur Verfügung gestellt.

Es gibt Präsenzübungen sowie drei kleine Programmierprojekte in C++. Wer die letzteren macht und bei mir einreicht sowie die Klausur am Ende besteht, bekommt die Vorlesung mit 5 LP angerechnet.

# 2 Mathematische Grundlagen

Wegen des oben Gesagten ist es sinnvoll, die wichtigsten Begriffe und Konzepte aus der linearen Algebra kurz zu wiederholen und zu interpretieren.

## 2.1 Notation

In dieser Vorlesung werden hauptsächlich reellwertige Vektoren und Matrizen betrachtet. Der Vektorraum ist also (fast) immer $\mathbb{R}^n$. Die Matrizen sind (fast) immer $\mathbb{R}^{m\times n}$. Die Skalare werden meist mit griechischen Buchstaben bezeichnet: $\alpha, \beta, \lambda$.... Die Vektoren sind Spaltenvektoren:

$$v = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix}.$$

und heißen meist $p, q, v, w$... oder auch mal $v_1, v_2$... (falls keine Verwechslungsgefahr besteht), oder $v^{(1)}, v^{(2)}, \ldots$ (falls doch).

Die wichtigsten Rechenoperationen für Vektoren (und Skalare) sind diese:

- Zahl mal Vektor:
$$\alpha v = \begin{pmatrix} \alpha v_1 \\ \alpha v_2 \\ \vdots \\ \alpha v_n \end{pmatrix}$$
- Vektor plus Vektor:
$$v + w = \begin{pmatrix} v_1 + w_1 \\ v_2 + w_2 \\ \vdots \\ v_n + w_n \end{pmatrix}$$
- Linearkombination von $m$ Vektoren $v^{(1)}, \ldots, v^{(m)}$
$$\sum_{i=1}^{m} \alpha_i v^{(i)}, \text{ bzw. } \alpha_1 v^{(1)} + \cdots + \alpha_m v^{(m)}$$
- Skalarpodukt (aka Innenprodukt):
$$\langle v, w \rangle = v^T \cdot w = v_1 \cdot w_1 + \cdots + v_n \cdot w_n = \sum_{i=1}^{n} v_i w_i$$
- Länge (oder Norm) eines Vektors:
$$\|v\| = \sqrt{v^T \cdot v} = \sqrt{\sum_{i=1}^{n} v_i^2}$$
- Winkel $\alpha$ zwischen $v$ und $w$:
$$\cos(\alpha) = \frac{\langle v, w \rangle}{\|v\| \|w\|}, \quad \text{also} \quad \alpha = \arccos \frac{\langle v, w \rangle}{\|v\| \|w\|}$$

Also ist der Kosinus des Winkel $\alpha$ zwischen zwei Vektoren $v, w$ der Länge 1 gerade das Skalarprodukt $v^T w$.

Matrizen werden mit lateinischen Großbuchstaben bezeichnet: $A, B \in \mathbb{R}^{m \times n}$. Dabei ist $m$ die Anzahl der Zeilen und $n$ die Anzahl der Spalten. Der Eintrag in Zeile $i$ und Spalte $j$ von $A$ heißt $a_{ij}$, oder auch $(A)_{ij}$. Die Matrix als Ganzes heißt ja $A$; aber manchmal ist auch praktisch, die Matrix als $(a_{ij})_{ij}$ zu schreiben. Also ist
$$A = (a_{ij})_{ij} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

Eine Matrix können wir natürlich auch als $n$ Spaltenvektoren nebeneinander auffassen.

Folgende grundlegende Operationen sind auf Matrizen $A, B \in \mathbb{R}^{m \times n}$ machbar.

- Multiplikation mit einem Skalar: $\alpha A$.
- Addition zweier Matrizen: $A+B$.
- Linearkombination: $\alpha A+\beta B+\cdots$.
- Matrix-Norm: später.
- Transponierte Matrix: $A^T$. Zeilen werden zu Spalten, Spalten zu Zeilen:

$$(A^T)_{ij} = (A)_{ji}$$

- Inverse Matrix $A^{-1}$.

Anders als Vektoren kann man Matrizen miteinander multilizieren, so dass wieder eine Matrix rauskommt. Genauer: $A$ aus $\mathbb{R}^{\ell\times m}$ und $B$ aus $\mathbb{R}^{m\times n}$, so kann man $A$ mal $B$ berechnen. Nennen wir das Ergebnis $C$, also $C = A\cdot B$, so ist

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{im}b_{mj}.$$

Ein Beispiel:

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \cdot \begin{pmatrix} 6 & -1 \\ 3 & 2 \\ 0 & -3 \end{pmatrix} = \begin{pmatrix} 1\cdot 6+2\cdot 3+3\cdot 0 & 1\cdot(-1)+2\cdot 2+3\cdot(-3) \\ 4\cdot 6+5\cdot 3+6\cdot 0 & 4\cdot(-1)+5\cdot 2+6\cdot(-3) \end{pmatrix} = \begin{pmatrix} 12 & -6 \\ 39 & -12 \end{pmatrix}$$

Im Allgemeinen ist $A\cdot B \neq B\cdot A$, auf mathematisch: Matrizenmultiplikation ist nicht *kommutativ*. Immerhin gilt $A\cdot(B\cdot C) = (A\cdot B)\cdot C$, also ist Matrizenmultiplikation *assoziativ*.

### Übersicht der zentralen Begriffe in Linearer Algebra:

Für Details siehe das Skript des Auffrischungskurses zu Mathe 1:

`https://www.math.uni-bielefeld.de/~frettloe/teach/alte-vorles/ueb/auff-skript-1-22.pdf`

- **Untervektorraum (UVR, auch: Unterraum):** eine Teilmenge eines Vektorraums, die selber Vektorraum ist. Formal ist $U \subset V$ ein UVR, falls
  - $0 \in U$ (das folgt auch aus Punkt 3, also eigtl überflüssig)
  - $\forall v,w \in U:\ v+w \in U$
  - $\forall \alpha \in \mathbb{R},\ v \in U:\ \alpha v \in U$
- **linear unabhängig:** die Vektoren $v_1,\ldots,v_m$ heißen linear unabhängig, falls sich keiner als Linearkombination der anderen darstellen lässt. Formal: Aus $\alpha_1 v_1 + \cdots + \alpha_n v_n = 0$ folgt $\alpha_1 = 0,\ldots,\alpha_m = 0$.
- **Linearkombination** (von Vektoren $v_1, v_2, \ldots, v_m$): jeder Ausdruck der Form

$$\alpha_1 v_1 + \alpha_2 v_2 + \cdots + \alpha_m v_m.$$

- **Spann** (aka lineare Hülle) von $v_1,\ldots,v_m$: Menge aller Linearkombinationen der $v_1,\ldots,v_m$.

- **Erzeugendensystem:** eine Menge $\{v_1, \ldots, v_m\}$ von Vektoren in einem (U)VR $V$, so dass sich jedes $v \in V$ als Linearkombination der $v_i$ schreiben lässt. Also formal: für jedes $v \in V$ gibt's $\alpha_1, \ldots, \alpha_m \in \mathbb{R}$ so dass
$$v = \alpha_1 v_1 + \cdots + \alpha_m v_m.$$
- **Basis:** linear unabhängiges Erzeugendensystem eines VR.
- **Dimension** (eines VR $V$): Zahl der Basisvektoren irgendeiner Basis von $V$.
- **Kern** (einer Matrix $A$, auch ker($A$)): Menge aller Vektoren, die von $A$ auf den Nullvektor abgebildet werden. Formal also $\{v \in \mathbb{R}^n \,|\, Av = 0\}$.
- **Bild** (einer Matrix $A$, auch im($A$)): Menge aller Vektoren, die als Ergebnis $Av$ auftreten. Formal also $\{w \,|\, w = Av \text{ für ein } v \in \mathbb{R}^n\}$.
- **Rang** (einer Matrix $A$): Dimension des Bildes von $A$.
- **lineare Abbildung:** Eine Abbildung $f : V \to W$ mit
  - $\forall u, v \in V$: $f(u+v) = f(u) + f(v)$.
  - $\forall v \in V, \alpha \in \mathbb{R}$: $f(\alpha v) = \alpha f(v)$.
- Zentrales Resultat: **Matrizen sind lineare Abbildungen und umgekehrt**.
  Genauer: jede lineare Abbildung $f : \mathbb{R}^m \to \mathbb{R}^n$ ist von der Form $f(x) = Ax$, mit $A \in \mathbb{R}^{m \times n}$; und jedes $f : \mathbb{R}^m \to \mathbb{R}^n$ mit $f(x) = Ax$ ist eine lineare Abbildung.
- **Inverse** (einer $n \times n$-Matrix $A$): die Matrix $A^{-1}$, so dass gilt: $A^{-1}A = E_n$ (falls es so ein $A^{-1}$ gibt).

## 2.2 Orthogonale Vektoren und Matrizen

Wir brauchen im Folgenden oft nur den Sonderfall, dass $v$ und $w$ senkrecht ("orthogonal") zueinander sind (also der Winkel $\pi/2$ ist).

**Definition 2.1.** Zwei Vektoren $v$ und $w$ sind **orthogonal** zueinander, wenn $v^T \cdot w = 0$.
Eine Menge $\{q_1, \ldots, q_k\}$ von Vektoren ist **orthogonal**, wenn alle Vektoren paarweise orthogonal sind (also, wenn $q_i^T \cdot q_j = 0$ für alle $1 \leq i < j \leq k$).
Eine Menge von Vektoren $\{q_1, \ldots, q_k\}$ ist **orthonormal**, wenn sie orthogonal ist und alle Vektoren Länge 1 haben (also, wenn $q_i^T \cdot q_i = 1$ für alle $1 \leq i \leq k$).

Es ist leicht zu sehen, dass eine Menge orthogonaler Vektoren linear unabhängig sein muss.

In Kapitel 8 brauchen wir den Begriff des **orthogonalen Komplements** $w^\perp$: ist $w \in \mathbb{R}^m$, dann ist das orthogonale Komplement alles, was senkrecht zu $w$ ist:

$$w^\perp = \{v \in \mathbb{R}^m \mid w^T \cdot v = 0\}$$

Man mache sich das im $\mathbb{R}^3$ für $w = (1,0,0)^T$ deutlich!)

Zusätzlich zu orthogonalen/orthonormalen Vektoren benötigen wir noch orthogonale Matrizen.

**Definition 2.2.** Eine Matrix $Q \in \mathbb{R}^{m \times m}$ heißt **orthogonal**, falls $Q^T Q = I$ ist.

Orthogonal sein heißt, dass die Spalten der Matrix alle orthonormal zueinander sind. (Dann sind es auch automatisch die Zeilen). Außerdem gilt für orthogonale Matrizen automatisch $QQ^T = I$.

Für Experten: Die Eigenwerte einer orthogonalen Matrix haben alle den (komplexen) Betrag 1.

Eine der wichtigsten Eigenschaften orthogonaler Matrizen ist, dass sie winkelerhaltend sind: Der Winkel zwischen $v$ und $w$ ändert sich nicht, wenn beide mit einer orthogonalen Matrix $Q$ multipliziert werden:

$$(Qv)^T \cdot (Qw) = v^T \cdot Q^T \cdot Qw = v^T \cdot w.$$

Da das Skalarprodukt auch die (quadrierte) Länge eines Vektors misst (siehe oben), werden Längen ebenso erhalten.

Die geometrische Interpretation der Winkel- und Längenerhaltung ist, dass orthogonale Matrizen ausschließlich Rotationen, Spiegelungen oder Kombinationen dieser beiden Transformationen darstellen können.

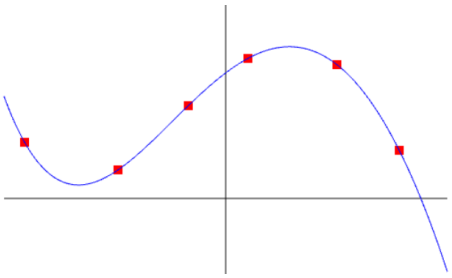
**Obacht:** Orthogonale Matrizen sind per Definition quadratisch. Gibt es auch nicht-quadratische Matrizen $A$, so dass $A^T \cdot A = I$? (Antwort in der Übung.)

# 3 Interpolation von Kurven

16. April

Als motivierendes Beispiel betrachten wir in diesem Kapitel das Problem der Kurveninterpolation. Hierbei wird das Aufstellen linearer Gleichungssysteme demonstriert und die Frage der Lösbarkeit solcher Systeme diskutiert.

Betrachten wir zunächst ein einfaches Interpolationsproblem: Finde eine Kurve $f : \mathbb{R} \to \mathbb{R}$, die die Werte $y_i$ an den Punkten $x_i$ für $i = 1, \ldots, m$ *interpoliert*; das heißt, es soll gelten $f(x_i) = y_i$ für $i = 1, \ldots, m$.



Kurveninterpolation findet bei zahlreichen Problemen Anwendung:

- 1D: Fitting von gemessenen Daten
- 2D: Zeichenprogramm
- 3D: Animationspfade, Kamerafahrten

Der Einfachheit halber beschränken wir uns zunächst auf den eindimensionalen Fall. Bei genauerer Betrachtung ist die oben formulierte Problemstellung noch nicht präzise genug: Es ist offen, ob nach genau *der* Kurve mit $f(x_i) = y_i$, oder nach irgendeiner Kurve mit dieser Eigenschaft gesucht werden soll. Wie man schnell sieht, gibt es unendlich viele Kurven mit $f(x_i) = y_i$ . Man benötigt also eine Methode, um den Lösungs- oder Suchraum einzuschränken.

## 3.1 Diskretisierung

Die grundlegende Idee bei der Diskretisierung ist, die Kurve mit einer endlichen Zahl von Parametern zu beschreiben. Die Kurve wird dazu mittels Basisfunktionen $p_j : \mathbb{R} \to \mathbb{R}$ und Koeffizienten $f_j \in \mathbb{R}$ dargestellt:

$$f(x) = \sum_{j=1}^{n} f_j p_j(x).$$

Als Basisfunktionen kommen dabei verschiedene Arten von Funktionen in Frage: Die Monom- Basis $x^i$ für Polynome, die Legendre-Polynome, die Tschebyscheff-Polynome, oder die Bernstein-Basisfunktionen $B_i^n(x)$ für polynomielle Bezier-Kurven, Spline-Basisfunktionen, oder sin/cos-Schwingungen für Fouriertransformationen. Die letzteren sehen wir vielleicht im zweiten Teil der Vorlesung.

In dieser Vorlesung betrachten wir ansonsten nur die Monome als Basispolynome, also

$$p_1(x) = x^0 = 1, p_2(x) = x^1, \ldots, p_n(x) = x^{n-1}.$$

Bezüglich dieser Basis wird $f$ dargestellt als

$$f(x) = \sum_{j=1}^{n} f_j \cdot x^{j-1} = f_1 + f_2 x + f_3 x^2 + \cdots + f_n x^{n-1}.$$

Da $f$ ein Polynom vom Grad $n-1$ ist, gibt es $n$ Freiheitsgrade $f_1, \ldots, f_n$ ("Degrees of Freedom", DoF). Aus der Diskretisierung ergibt sich somit eine veränderte diskrete Problemstellung: Finde Koeffizienten $f_1, \ldots, f_n$, so dass

$$f(x_i) = \sum_{j=1}^{n} f_j \cdot x_i^{j-1} = y_i, \quad i = 1, \ldots, m.$$

Daraus ergeben sich $m$ Gleichungen mit $n$ Unbekannten:

$$\begin{array}{rccccccccc}
 & f_1 x_1^0 & + & f_2 x_1^1 & + & \cdots & + & f_n x_1^{n-1} & = & y_1, \\
\wedge & f_1 x_2^0 & + & f_2 x_2^1 & + & \cdots & + & f_n x_2^{n-1} & = & y_2, \\
 & \vdots & & \vdots & & & & \vdots & & \vdots \\
\wedge & f_1 x_m^0 & + & f_2 x_m^1 & + & \cdots & + & f_n x_m^{n-1} & = & y_m.
\end{array}$$

Da die Funktionswerte $f(x_i)$ linear von den Koeffizienten $f_1, \ldots, f_n$ abhängen, können die $m$ Gleichungen zu einem linearen Gleichungssystem zusammengefasst werden:

$$\begin{pmatrix} 1 & x_1 & \cdots & x_1^{n-1} \\ \vdots & & & \vdots \\ 1 & x_m & \cdots & x_m^{n-1} \end{pmatrix} \cdot \begin{pmatrix} f_1 \\ \vdots \\ f_n \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix}.$$

## 3.2 Allgemeine Formulierung

Mit einer allgemeinen Polynombasis $\{p_1, \ldots, p_n\}$ sieht das so aus:

$$f(x_i) = \sum_{j=1}^{n} f_j \cdot p_j(x_i) = y_i, \quad i = 1, \ldots, m.$$

Daraus ergeben sich $m$ Gleichungen mit $n$ Unbekannten:

$$\begin{array}{ccccccccc} f_1p_1(x_1) & + & f_2p_2(x_1) & + & \cdots & + & f_np_n(x_1) & = & y_1, \\ f_1p_1(x_2) & + & f_2p_2(x_2) & + & \cdots & + & f_np_n(x_2) & = & y_2, \\ \vdots & & \vdots & & & & \vdots & & \vdots \\ f_1p_1(x_m) & + & f_2p_2(x_m) & + & \cdots & + & f_np_n(x_m) & = & y_m. \end{array}$$

Da die Funktionswerte $f(x_i)$ immer noch linear von den Koeffizienten $f_1,\ldots,f_n$ abhängen, können die $m$ Gleichungen wieder zu einem linearen Gleichungssystem zusammengefasst werden:

$$\begin{pmatrix} p_1(x_1) & \cdots & p_n(x_1) \\ \vdots & & \vdots \\ p_1(x_m) & \cdots & p_n(x_m) \end{pmatrix} \cdot \begin{pmatrix} f_1 \\ \vdots \\ f_n \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix}, \text{ oder kurz:}$$

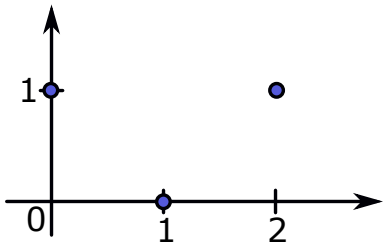
$$A \cdot x = b, \quad \text{mit } A \in \mathbb{R}^{m\times n}, x \in \mathbb{R}^n, b \in \mathbb{R}^m \tag{3.1}$$

Dies führt zu der Frage, wann ein Gleichungssystem $Ax = b$ (eindeutig) lösbar ist. Dafür ist die Dimension der Matrix $A$ von entscheidender Bedeutung:

- Wenn $m = n$: $A$ ist quadratisch. Wenn $A$ zudem vollen Rang hat, gibt es eine inverse Matrix $A^{-1}$, so dass eine eindeutige Lösung $x = A^{-1}b$ existiert.
- Wenn $m > n$: Das System hat mehr Gleichungen als Unbekannte, es ist *überbestimmt* und daher im Allgemeinen nicht exakt lösbar. (Fast immer.)
- Wenn $m < n$: Das System hat zu viele Unbekannte, es ist *unterbestimmt*. Es gibt keine eindeutige Lösung, sondern einen ganzen Raum von möglichen Lösungen. (Praktisch immer, bis auf "dumme" Ausnahmen wie $x_1 + x_2 + x_3 + x_4 = 0 \wedge x_4 = 1 \wedge x_4 = 2$.)

## 3.3 Ein Spielzeugbeispiel

Um die Dinge oben zu illustrieren betrachten wir das Beispiel $f(0) = 1$, $f(1) = 0$, $f(2) = 1$. (Also $(x_0, y_0) = (0,1)$, $(x_1, y_1) = (1,0)$, $(x_2, y_2) = (2,1)$.)




Für $m = 2, n = 2$ ergibt sich das LGS

$$\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} f_1 \\ f_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

Die eindeutige Lösung ist $f_1 = 1, f_2 = -1$. Also ist $f(x) = 1 - x$, siehe Abbildung 1 links.

Für $m = 3, n = 2$ ergibt sich das LGS

$$\begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \end{pmatrix} \cdot \begin{pmatrix} f_1 \\ f_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}.$$

Das LGS ist überbestimmt und hat keine Lösung. (Klar, es gibt kein Polynom vom Grad $n-1=1$, dass alle drei Punkte trifft).

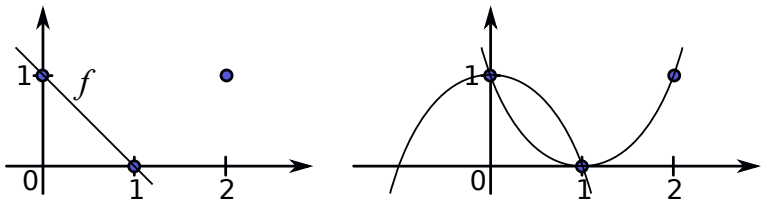



Figure 1: Für $m = n = 2$ ergibt sich die Lösung $f_1 = 1, f_2 = -1$, also das eindeutige Interpolationspolynom $x-1$ (links). Für $m = 2, n = 3$ sind hier zwei Lösungen (von unendlich vielen) gezeigt, $x^2-1$ und $x^2-2x+1$.

Für $m = 2, n = 3$ ergibt sich das LGS

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

Das LGS ist unterbestimmt. Die (unendlich vielen) Lösungen sind $f_1 = 1, f_2$ beliebig, $f_3 = -1 - f_2$. $f(x) = 1 - x$, siehe Abbildung 1 rechts.

### 3.4 Der Fall $m = n$

Die Frage, wann eine quadratische $(m \times m)$-Matrix vollen Rang hat, ist von entscheidender Bedeutung für die Lösung des linearen Gleichungssystems. Betrachten wir dazu die Matrix für die Monombasis noch einmal genauer:

$$A = \begin{pmatrix} 1 & x_1 & x_1^2 & \dots & x_1^{m-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{m-1} \\ 1 & x_3 & x_3^2 & \dots & x_3^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & x_m^2 & \dots & x_m^{m-1} \end{pmatrix}. \tag{3.2}$$

Die Matrix hat keinen vollen Rang, wenn: $x_i = x_j$ für irgendwelche $i \neq j$: die Zeilen $i$ und $j$ sind dann identisch. Also kein voller Rang, also keine Inverse, Matrix singulär.

Daraus folgt, dass die $x_i$ unterschiedlich sein müssen. Man kann zeigen, dass es bei unterschiedlichen $x_i$ ein eindeutiges Polynom vom Grad $\leq m-1$ gibt, das die Werte $y_i$ an den Punkten $x_i$ interpoliert. Wir skizzieren den Beweis hier nur kurz:

Seien $f, g$ Polynome vom Maximalgrad $m-1$ mit $f(x_i) = g(x_i) = y_i$ für $i = 1, \dots, m$. Dann ist auch $h = (f-g)$ ein Polynom vom Maximalgrad $m-1$. Aber $h$ hat $m$ Nullstellen $x_1, x_2, \dots, x_m$. Das geht nur für $h = 0$, also $f = g$.

Die Tatsache, dass es ein eindeutiges Polynom gibt, bedeutet, dass es eine eindeutige Lösung des linearen Gleichungssystems gibt. Daher muss die $m \times m$-Matrix $A$ vollen Rang haben. Beweisskizze Ende.

Man kann mit linearer Algebra zeigen, dass

$$\det(A) = \prod_{0 \le i < j \le n} (x_j - x_i).$$

Das ist aufwändiger, siehe wikipedia. Insbesondere erhält man aber, dass

$$\forall i \neq j : x_i \neq x_j \;\Leftrightarrow\; \det(A) \neq 0 \;\Leftrightarrow\; A \text{ hat vollen Rang.}$$

# 4 LU-Zerlegung

Für die Polynominterpolation oben müssen wir nun also ein LGS lösen. Im Prinzip können wir das ja: erstes Semester, Zeilen-Stufen-Form herstellen, bzw auf vornehm: Gaußsches Eliminationsverfahren. Wie bringen wir das einem Computer bei? Dazu benutzen wir die sogenannte LU-Zerlegung (für *lower* und *upper*, aka LR-Zerlegung, für links und rechts).

Sei $Ax = b$ mit $A \in \mathbb{R}^{m \times m}$ und $b \in \mathbb{R}^m$. Gesucht ist ja $x \in \mathbb{R}^m$ Der Trick ist, zwei Matrizen $L, U$ mit $A = L \cdot U$ zu finden, so dass $L$ eine untere Dreiecksmatrix (*lower*) ist und $U$ eine obere Dreiecksmatrix *(upper)*. Dann ist ja

$$Ax = b \quad \Leftrightarrow \quad LUx = b.$$

Nun lässt sich das ursprüngliche Gleichungssystem in zwei Teilprobleme aufteilen: (1) Löse $Ly = b$ und (2) löse $Ux = y$. Das ist jeweils einfach, weil $L$ und $U$ ja jeweils schon Dreiecksgestalt (Zeilen-Stufen-Form!) haben. Ein solches Gleichungssystem lässt sich leicht durch schrittweise Substitution lösen. Sei $L \in \mathbb{R}^{m \times m}$ eine untere Dreiecksmatrix und $y, b \in \mathbb{R}^m$:

$$\begin{pmatrix} \ell_{11} & 0 & \cdots & 0 \\ \ell_{21} & \ell_{22} & \ddots & \vdots \\ \vdots & & \ddots & 0 \\ \ell_{m1} & \cdots & \cdots & \ell_{mm} \end{pmatrix} \cdot \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix}$$

- Ausschreiben der ersten Zeile liefert $l_{11} y_1 = b_1$, also $y_1 = \frac{b_1}{\ell_{11}}$.
- Ausschreiben der zweiten Zeile: $\ell_{21} y_1 + \ell_{22} y_2 = b_2$, also $y_2 = \frac{1}{\ell_{22}}(b_2 - \ell_{21} y_1)$.
- Generell kann nach Bearbeiten der ersten $(i-1)$ Zeilen die Unbekannte $y_i$ aus der $i$-ten Zeile und den nun bekannten $y_1, \ldots, y_{i-1}$ wie folgt berechnet werden:

$$y_i = \frac{1}{\ell_{ii}} \Big( b_i - \sum_{j=1}^{i-1} \ell_{ij} y_j \Big) \quad \text{für } i = 1, 2, \ldots, m.$$

Diese Vorgehensweise wird auch Vorwärts-Substitution genannt. Das ist schon sehr ähnlich dem, was wir beim Berechnen eines LGS von Hand tun. Das nächste ist exakt das, was wir beim Lösen eines LGS von Hand tun.

Sei also nun $U \in \mathbb{R}^{m\times sm}$ eine obere Dreiecksmatrix und $x, y \in \mathbb{R}^m$.

$$\begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1m} \\ 0 & u_{22} & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & u_{mm} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ \vdots \\ x_m \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix}$$

Dann können die $x_i$ von unten nach oben berechnet werden:

- Aus der letzten Zeile folgt $x_m = \frac{y_m}{u_{mm}}$.
- Dann kann $x_i$ aus der $i$-ten Zeile und den schon bekannten $x_{i+1}, \ldots, x_m$ wie folgt berechnet werden:
$$x_i = \frac{1}{u_{ii}}\Big(y_i - \sum_{j=i+1}^{m} u_{ij}x_j\Big) \quad \text{für } i = m-1, \ldots, 1.$$

Diese Vorgehensweise wird als Rückwärts-Substitution bezeichnet.

Bei der Gauß-Elimination wird die $(m \times m)$-Matrix $A$ nach und nach in eine obere Dreiecksmatrix $U$ transformiert. Wir werden nun sehen, dass die Transformationen, die wir dabei durchführen, die Matrix $L$ liefern. 23. April

Denn: es werden, von Spalte 1 bis Spalte $m$, schrittweise Nullen unterhalb der Haupt-Diagonalen eingeführt. Dies geschieht durch das Subtrahieren von Vielfachen der $i$-ten Zeile von den unteren Zeilen $i+1, \ldots, m$. Das lässt sich durch die Multiplikation mit einer einfachen unteren Dreiecksmatrix $L_i$ beschreiben.

Bei einer $4 \times 4$-Matrix sieht das Ganze ja typischerweise so aus:

$$\begin{pmatrix} \star & \star & \star & \star \\ \star & \star & \star & \star \\ \star & \star & \star & \star \\ \star & \star & \star & \star \end{pmatrix} \to \begin{pmatrix} \star & \star & \star & \star \\ 0 & \star & \star & \star \\ 0 & \star & \star & \star \\ 0 & \star & \star & \star \end{pmatrix} \to \begin{pmatrix} \star & \star & \star & \star \\ 0 & \star & \star & \star \\ 0 & 0 & \star & \star \\ 0 & 0 & \star & \star \end{pmatrix} \to \begin{pmatrix} \star & \star & \star & \star \\ 0 & \star & \star & \star \\ 0 & 0 & \star & \star \\ 0 & 0 & 0 & \star \end{pmatrix}$$

$$A \qquad L_1A \qquad L_2L_1A \qquad L_3L_2L_1A$$

Die resultierende Matrix auf der rechten Seite ist dann die gewünschte obere Dreiecksmatrix $U$. Mit $L' = L_3L_2L_1$ ist also $L'A = U$, also mit $L := (L_3L_2L_1)^{-1}$ ist $A = LU$. Bingo. Obwohl: Zunächst sieht das mit der Inversen ja abschreckend aus. Wir werden uns im Folgenden aber überzeugen, dass dies Inverse sehr leicht zu berechnen ist. Zunächst aber ein konkretes Zahlenbeispiel zur Illustration.

**Beispiel 4.1.** Wir nehmen

$$A = \begin{pmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{pmatrix}$$

Im ersten Schritt werden die drei Einträge in der ersten Spalte unter der 2 zu 0 gemacht. Als Matrixoperation sieht das so aus:

$$L_1A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -2 & 1 & 0 & 0 \\ -4 & 0 & 1 & 0 \\ -3 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{pmatrix} = \begin{pmatrix} 2 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 3 & 5 & 5 \\ 0 & 4 & 6 & 8 \end{pmatrix}.$$

Also $-2$mal die erste Zeile plus die zweite usw. Schritt 2 ist dann:

$$L_2L_1A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -3 & 1 & 0 \\ 0 & -4 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 3 & 5 & 5 \\ 0 & 4 & 6 & 8 \end{pmatrix} = \begin{pmatrix} 2 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 2 & 2 \\ 0 & 0 & 2 & 4 \end{pmatrix}.$$

Und zuletzt:

$$L_3L_2L_1A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{pmatrix} \begin{pmatrix} 2 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 2 & 2 \\ 0 & 0 & 2 & 4 \end{pmatrix} = \begin{pmatrix} 2 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & 2 \end{pmatrix} = U.$$

Man beachte, dass die Matrix $U$ genau das ist, was das Gaussverfahren uns als Zeilen-Stufen-Form liefert. Wir brauchen noch das $L$:

$$L = (L_3L_2L_1)^{-1} = L_1^{-1}L_2^{-1}L_3^{-1}.$$

Das sieht zunächst abschreckend aus. Es zeigt sich aber, dass aufgrund der sehr speziellen Form der $L_i$ die Inversen und Produkte sehr leicht zu berechnen sind (siehe Übungsblatt 2). Z.B. ist

$$L_1^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -2 & 1 & 0 & 0 \\ -4 & 0 & 1 & 0 \\ -3 & 0 & 0 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 4 & 0 & 1 & 0 \\ 3 & 0 & 0 & 1 \end{pmatrix}.$$

Das ist ein allgemeines Prinzip: Um die Inverse $L_i^{-1}$ von $L_i$ zu erhalten, ändert man einfach alle Vorzeichen der Einträge unter der Hauptdiagonale.

Genau so einfach (wieder aufgrund der speziellen Form der $L_i$) ist das Ausmultiplizieren: Im Beispiel ist

$$L = L_1^{-1}L_2^{-1}L_3^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 4 & 3 & 1 & 0 \\ 3 & 4 & 1 & 1 \end{pmatrix}.$$

Das Produkt ergibt sich also einfach durch Kopieren-und-Einfügen der Einträge der $L_i^{-1}$. (Obacht: die Reihenfolge ist wichtig! Z.B. sieht $(L_1L_2L_3)^{-1} = L_3^{-1}L_2^{-1}L_1^{-1}$ ganz anders aus und lässt sich nicht durch Kopieren-und-Einfügen berechnen!)

## 4.1 Der LU-Algorithmus

Sei $A^{(k)}$ die Matrix $L_{k-1}\cdots L_1A$, die sich nach $k-1$ Iterationen der Gauß-Elimination ergibt. Die Einträge von $A_k$ sind $a_{ij}^{(k)}$. Dann ergibt sich $L_k$ als

$$L_k = \begin{pmatrix} 1 & & & & \\ & \ddots & & & \\ & & 1 & & \\ & & -\ell_{k+1,k} & 1 & \\ & & \vdots & \ddots & \\ & & -\ell_{m,k} & & 1 \end{pmatrix} \quad \text{mit } \ell_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}, \quad i = k+1, \ldots, m. \tag{4.1}$$

Die Inversen $L_k^{-1}$ und die Produkte derselben werden wie im letzten Beispiel berechnet.

Fasst man alle Schritte kompakt zusammen, ergibt sich der Pseudo-Code in Algorithmus 1 für die LU-Zerlegung (ohne Pivotisierung, dazu später). Dabei werden für die Matrizen $L$ und $U$ jeweils nur die relevanten Einträge gespeichert (für $L$: $\ell_{ij}$ mit $i > j$, für $U$: $u_{ij}$ mit $i \leq j$). Intern könnten also auch beide Matrizen in einer einzelnen Matrix (z.B. $A$) gespeichert werden.

**Algorithm 1 LU-Zerlegung ohne Pivotisierung**. Eingabe: Matrix $A$. Ausgabe: Matrix $L$, Matrix $U$.

```
U = A
for k = 1...m-1 do
  for i = k+1...m do
    l[i,k] = u[i,k]/u[k,k]
    for j = k...m do
      u[i,j] = u[i,j] - l[i,k]u[k,j]
    end for
  end for
end for
```

## 4.2 Stabilität

Vielleicht haben Sie es schon bemerkt: In der bisher dargestellten Form ist die Gauß-Elimination nicht zur Lösung allgemeiner linearer Gleichungssysteme geeignet. Für bestimmte Matrizen scheitert der Algorithmus gänzlich, da es zu einer Division durch Null kommen kann. Bei der Matrix

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

würde die Gauß-Elimination bereits im ersten Schritt fehlschlagen. Weitere Schwierigkeiten ergeben sich durch die Gleitkomma-Arithmetik. Die Gauß-Elimination der Matrix

$$A = \begin{pmatrix} \varepsilon & 1 \\ 1 & 1 \end{pmatrix}$$

ergibt formell die LU-Zerlegung

$$A = L \cdot U = \begin{pmatrix} 1 & 0 \\ \frac{1}{\varepsilon} & 1 \end{pmatrix} \cdot \begin{pmatrix} \varepsilon & 1 \\ 0 & 1 - \frac{1}{\varepsilon} \end{pmatrix}$$

Führt man die LU-Zerlegung allerdings numerisch durch und wählt $\varepsilon$ als sehr kleine Zahl, z.B. $10^{-20}$, ergeben sich Abweichungen. Das Ergebnis der Zerlegung ist dann

$$\tilde{L} \cdot \tilde{U} = \begin{pmatrix} 1 & 0 \\ 10^{20} & 1 \end{pmatrix} \cdot \begin{pmatrix} 10^{-20} & 1 \\ 0 & -10^{20} \end{pmatrix}$$

Im Eintrag $\tilde{u}_{22}$ ergibt sich $-10^{20}$ anstatt des korrekten Ergebnisses $1 - 10^{20}$. Auch wenn der Fehler auf den ersten Blick sehr klein erscheint, kann er dennoch schwerwiegende Folgen haben. Betrachten wir dazu die Matrix

$$\tilde{A} = \tilde{L} \cdot \tilde{U} = \begin{pmatrix} 10^{-20} & 1 \\ 1 & 0 \end{pmatrix}$$

Im Vergleich zur ursprünglichen Matrix $A$ sollte der Eintrag $\tilde{a}_{22}$ eigentlich 1 und nicht 0 sein. Der kleine Fehler in $\tilde{U}$ führt also zu einem großem Fehler in $\tilde{A}$. Die Lösung des Gleichungssystems $\tilde{L}\tilde{U}x = b$ würde sich gravierend von der Lösung von $Ax = b$ unterscheiden. Zum Beispiel wäre für $b = (1-\varepsilon, 0)^T$ die korrekte Lösung $x = (-1, 1)^T$, das numerische Resultat ergibt allerdings $\tilde{x} = (1-\varepsilon, 0)^T$. Es stellt sich also die Frage, warum aus $1 - 10^{20}$ plötzlich $-10^{20}$ wird. Zur Beantwortung dieser Frage ist es nötig, sich zunächst eingehender mit der Arithmetik von Gleitkommazahlen zu befassen.

## 4.3 Gleitkommaarithmetik

Gleitkommazahlen werden im Rechner mittels Vorzeichen $S$, Mantisse $M$, Basis $B$ und Exponent $E$ wie folgt dargestellt:

$$S \cdot M \cdot B^E$$

Betrachten wir folgendes Beispiel für den Fall $B = 10$:

$$123 = 123 \cdot 10^0 = 1,23 \cdot 10^2 = 12300 \cdot 10^{-2}.$$

Die Darstellung der Zahl ist offenbar nicht eindeutig. Abhilfe schafft dabei die normalisierte Darstellung. Man einigt sich auf:

1. Die erste Ziffer von $M$ muss ungleich 0 sein.
2. Das Komma muss nach der ersten Ziffer folgen.

Beispiele für normalisierte Darstellungen sind $1,23 \cdot 10^2$ und $4,56 \cdot 10^{-3}$. Während der Mensch üblicherweise mit dem Dezimalsystem ($B = 10$) arbeitet, werden Zahlen im Rechner meistens im Binärsystem ($B = 2$) dargestellt:

$$1,0101 \cdot 2^2 = 101,01 \cdot 2^0 = 10101 \cdot 2^{-2} \quad (= 5,25 \text{ in dezimal.}$$

Bei der normalisierten Darstellung im Binärsystem ist das erste Bit von $M$ immer 1 und muss dementsprechend nicht explizit gespeichert werden. (Quizfrage: wie speichere ich dann eine 0?)

Die Darstellung von Zahlen im Rechner ist diskret. Es gibt eine feste Anzahl von Ziffern für die Mantisse $M$ und den Exponenten $E$. Nach dem IEEE Standard for Floating-Point Arithmetic (IEEE 754) sind die Datentypen für einfache und doppelte Präzision (float und double in C++) aufgebaut wie in Abbildung 2.

Die Anzahl der Ziffern der Mantisse bestimmt dabei die Genauigkeit, die des Exponenten die (betragsmäßig) kleinste bzw. größte darstellbare Zahl. Der Exponent ist *biased*, soll heißen: eigentlich nimmt er Werte von 0 bis 255 an. Aber da wir ja auch kleine Zahlen wie $0,00000125$ speichern wollen, brauchen wir auch negative Exponenten. Daher wird der Wertebereich als $-127, ...128$ aufgefasst ($00000000 = -127, 00000001 = -126$ usw). Dabei sind aber $-127$ (nur Nullen) und 128 (nur Einsen) Sonderfälle. Der Wertebereich ist also $-126 \leq E \leq 127$.

Die Frage nach der Genauigkeit von float und double ist tricksig. Natürlich kommt es auf die Größe der Zahl an: selbst eine natürliche Zahl $n \in \mathbb{N}$ wird nicht mehr genau als float oder double dargestellt wenn sie nur genügend groß ist (vgl. Übung). Die Sonderfälle (wikipedia: subnormal numbers) machen die Frage nach der Genauigkeit noch viel komplizierter.

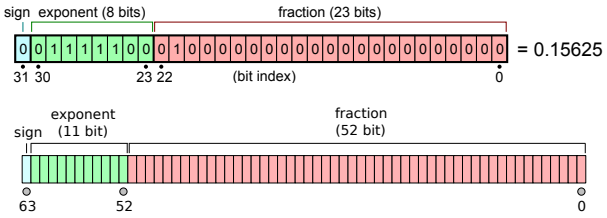



Figure 2: Gespeicherte Binärziffern bei `float` (oben) und `double` (unten). Das float zeigt die Zahl $\frac{5}{32} = \frac{4+1}{32} = 0,15625$. In der Mantisse steht (implizit) binär 101, das wird gelesen als $1,01$, also $1+\frac{1}{4}$, also $\frac{5}{4}$. Im Exponenten steht $127-3=124$. Das wird interpretiert als $2^{-3} = \frac{1}{8}$. Passt.

Ein Maß für die Genauigkeit ist das Maschinen-Epsilon: $\varepsilon_m$ ist die Differenz zwischen 1 und der nächstgrößeren Gleitkommazahl. Bei $p$ Ziffern für die Mantisse $M$ (unter Berücksichtigung der impliziten 1 vorne) und der Basis $B$ ergibt sich $\varepsilon_m$ als

$$1,\underbrace{0...01}_{p}\cdot B^0 - 1,\underbrace{00...0}_{p}\cdot B^0 = \underbrace{0,0...0}_{p}1\cdot B^0 = 1\cdot B^{-p} =: \varepsilon_m.$$

Praktischer Hinweis: In C++ können die Eigenschaften der verschiedenen numerischen Zahlentypen mittels `numeric_limits<xxx>` ermittelt werden. Für float kann z.B. mittels `numeric_limits<float>::epsilon()` das Maschinen-Epsilon für float-Typen abgefragt werden. Ebenso existieren die Funktionen `numeric_limits<xxx>::min()` und `numeric_limits<xxx>::max()`, um die kleinste und größte darstellbare Zahl abzufragen.

Im Folgenden betrachten wir die Addition zweier Gleitkommazahlen genauer:

$$M_1\cdot B^{E_1} + M_2\cdot B^{E_2}, \text{ mit o.B.d.A. } E_1 \geq E_2.$$

Der Ablauf ist wie folgt:

1. Die kleinere Zahl auf gleichen Exponenten bringen:
$$E_2' = E_2 + (E_1 - E_2) = E_1,$$
$$M_2' = M_2 / B^{(E_1-E_2)}.$$
2. Die Mantissen addieren: $M = M_1 + M_2'$.
3. Das Resultat $M\cdot B^{E_1}$ normalisieren und runden.

**Beispiel 4.2.** Dezimalbasis $B=10$ und $p=7$ Ziffern für $M$. Zu addieren sind die Zahlen $123456,7$ und $101,7654$:

$$1,234567\cdot 10^5 + 1,017654\cdot 10^2$$

Auf den gleichen Exponenten bringen:

$$1,234567\cdot 10^5 + 0,001017654\cdot 10^5 = 1,235584654\cdot 10^5$$

Da die Mantisse nur sieben Ziffern hat ($p=7$), ist das Ergebnis im Rechner $1,235585\cdot 10^5$. Die letzten drei Ziffern 6, 5 und 4 gehen verloren.

Es sind insbesondere solche Operationen von dieser Art Fehler betroffen, bei denen der Unterschied im Exponenten sehr groß ist. Die Addition einer sehr kleinen und einer sehr großen Zahl ist ein Beispiel. Auch im vorherigen Beispiel zur LU-Zerlegung war dies die Ursache für das falsche Ergebnis ($1 - 10^{20}$ wird zu $-10^{20}$).

## 4.4 LU-Zerlegung mit Pivotisierung

30. April

Durch eine einfache Modifikation der in Algorithmus 1 beschriebenen Gauß-Elimination kann die Instabilität der LU-Zerlegung kontrolliert werden. Die Idee besteht darin, durch Permutation der Zeilen der Matrix $A$ fehleranfällige Rechenoperationen zu vermeiden. Dieses Verfahren wird auch Pivotisierung genannt. Betrachten wir den $k$-ten Schritt von Algorithmus 1:

$$\begin{pmatrix} \star & \star & \star & \star \\ & a_{kk}^{(k)} & \star & \star \\ & \star & \star & \star \\ & \star & \star & \star \end{pmatrix} \quad \rightarrow \quad \begin{pmatrix} \star & \star & \star & \star \\ & a_{kk}^{(k)} & \star & \star \\ & 0 & \star & \star \\ & 0 & \star & \star \end{pmatrix}$$

$$L_{k-1} \cdots L_1 A \qquad\qquad L_k \cdots L_1 A$$

Die Matrizen $L_k$ sind definiert wie in (4.1), d.h., mit Subdiagonal-Einträgen $\ell_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}$ in der $k$-ten Spalte. Wenn die $a_{kk}^{(k)}$ sehr klein sind, werden die $\ell_{ik}$ sehr groß. Durch die Gleitkomma-Arithmetik ergeben sich dann Fehler bei der Gauß-Elimination. Der Ansatz bei der Pivotisierung ist, die Zeilen von $A$ zu vertauschen, um ein numerisch stabileres Diagonal-Element zu finden. Der Ablauf ist wie folgt:

1. Finde $i \in \{k, \ldots, m\}$, so dass $|a_{ik}^{(k)}|$ maximal ist.
2. Vertausche die Zeilen $i$ und $k$ mit einer Permutationsmatrix $P_k$ (s.u.).
3. Elimination mittels Matrix $L_k$.

Diese Schritte sind in folgender Abbildung verdeutlicht:

$$\begin{pmatrix} \star & \star & \star & \star \\ & a_{kk}^{(k)} & \star & \star \\ & \star & \star & \star \\ & a_{ik}^{(k)} & \star & \star \end{pmatrix} \quad \rightarrow \quad \begin{pmatrix} \star & \star & \star & \star \\ & a_{ik}^{(k)} & \star & \star \\ & \star & \star & \star \\ & a_{kk}^{(k)} & \star & \star \end{pmatrix} \quad \rightarrow \quad \begin{pmatrix} \star & \star & \star & \star \\ & a_{kk}^{(k)} & \star & \star \\ & 0 & \star & \star \\ & 0 & \star & \star \end{pmatrix}$$

$$A^{(k)} \qquad\qquad P_k A^{(k)} \qquad\qquad L_k P_k A^{(k)}$$

**Obacht:** Zum Lösen von $Ax = b$ berechnen wir nun also eine Lösung von $PAx = Pb$. Das ist wichtig in der Implementation!

Man spricht von partieller Pivotisierung, wenn nur innerhalb der Teilspalte $a_{k,k}^{(k)}, a_{k+1,k}^{(k)}, \ldots, a_{m,k}^{(k)}$ der betragsgrößte Eintrag gesucht wird. Sucht man dagegen innerhalb der ganzen Untermatrix unterhalb und rechts von $a_{kk}$, also in $(a_{i,j})_{k \le i \le m, k \le j \le n}$, spricht man von voller Pivotisierung. Die Permutationsmatrix $P$ ist überall 0 abgesehen von einer 1 pro Spalte und Zeile. Sie wird konstruiert, indem man mit der Einheitsmatrix $E$ startet und anschließend Zeilen vertauscht. In der Übung sahen wir bereits: Für eine Permutationsmatrix $P$ gilt: $P \cdot A$ vertauscht Zeilen von $A$, $A \cdot P$ vertauscht Spalten von $A$, und $P^{-1} = P^T$.

### 4.5 Rechenaufwand

Betrachten wir den Rechenaufwand für das Lösen von $Ax = b$ mittels LU-Faktorisierung. Die Anzahl der Operationen sind in FLoating point OPerations (FLOP) angegeben. Es sind drei Arbeitsschritte zu betrachten:

1. Faktorisierung $PA = LU$: Die dreifach geschachtelte for-Schleife im Algorithmus deutet auf kubischen Aufwand $O(m^3)$ hin. Zählt man genauer, erhält man $\frac{2}{3}m^3$ FLOP für die Gauß-Elimination.
2. Lösen von $Ly = Pb$: $O(m^2)$ FLOP für die Vorwärts-Substitution.
3. Lösen von $Ux = y$: $O(m^2)$ FLOP für die Rückwärts-Subsitution.

Die LU-Zerlegung kann auf verschiedene Weise optimiert werden. Wir erwähnen hier zwei Ansätze nur kurz:

- Genaues Hinsehen liefert, dass wir die $a_{ij}$ mit den $\ell_{ij}$ und $u_{ij}$ überschreiben können: Die benutzten $a_{ij}$ werden später nicht mehr gebraucht. Also können wir alles in *einer* Matrix $A$ speichern (Algorithmus von Crout). Dadurch reduziert sich insbesondere bei großen Matrizen der Speicherverbrauch deutlich (Faktor 2).
- Das $P$ hat als Matrix sehr viele Nullen, das speichern wir daher als Permutations-Array.

## 5 Kurven-Approximation

Bei der Kurven-Interpolation kommt es in verschiedenen Situationen zu Oszillationen der Interpolationskurve. Diese Schwankungen sind oftmals so groß, dass die Kurve keine sinnige Repräsentation der interpolierten Daten mehr ist (sogenanntes over-fitting). Ist z.B. die Anzahl der Interpolationspunkte — und dem entsprechend der Polynomgrad — zu hoch, so kommt es zu starken Schwankungen der Kurve. Abbildung 3 zeigt ein Beispiel für ein Polynom vom Grad 7. Auch bei wenigen Interpolationspunkten ist das Verhalten der Kurven nicht optimal: Insbesondere an den Enden des Intervalls treten Oszillationen auf, wie Abbildung 3 zeigt.

Ein weiteres Problem der Kurven-Interpolation ist die Empfindlichkeit gegenüber verrauschten Daten. Überlagert man die Interpolationspunkte mit einem Rauschen, so wie es bei realen Messdaten etwa der Fall ist, führt dies ebenso zu starken Schwankungen. Eine bessere Möglichkeit zum Fitting von Daten ist die Approximation mit einem Polynom niedrigeren Grades.

Wir wollen also Messpunkte $\{(x_1, y_i), \ldots, (x_m, y_m)\}$ mit großem $m$ durch ein Polynom $f(x) = \sum_{j=1}^{n} f_j x^{j-1}$ mit kleinem Grad $n < m$ approximieren. Das liefert das überbestimmte LGS

$$\underbrace{\begin{pmatrix} 1 & x_1 & x_1^2 & \ldots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \ldots & x_2^{n-1} \\ 1 & x_3 & x_3^2 & \ldots & x_3^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_m & x_m^2 & \ldots & x_m^{n-1} \end{pmatrix}}_{A} \cdot \underbrace{\begin{pmatrix} f_1 \\ \vdots \\ f_n \end{pmatrix}}_{x} = \underbrace{\begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix}}_{b}.$$

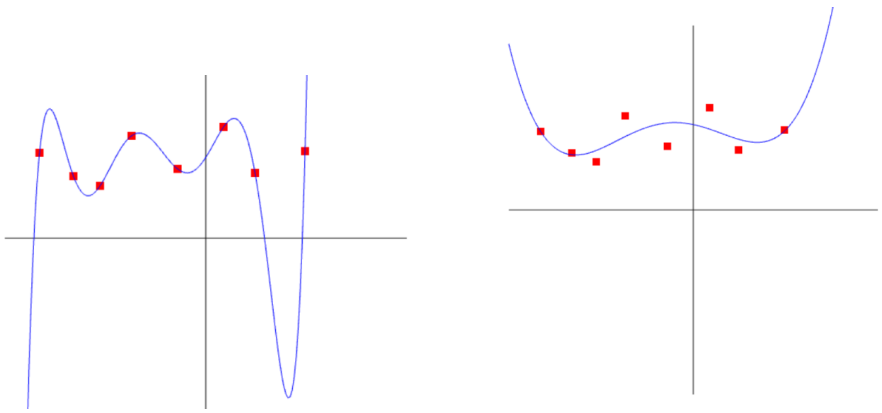


Figure 3: Interpolation (links) versus Approximation. Bei der Interpolation werden die Punkte genau getroffen, aber dazwischen oszilliert der Graph stark. Bei der Approximation verzichtet man auf die Forderung, die Punkte genau zu treffen. Dadurch (a) schwankt die Kurve nicht so stark, und (b) wir kommen mit einem Polynom von niedrigerem Grad aus. (Links ist der Grad 7, rechts 4.)

Die Matrix $A$ hat mehr Zeilen als Spalten. Das Gleichungssystem $Ax = b$ hat keine exakte Lösung, da $A$ nicht invertierbar ist. Wir wollen daher versuchen, das Gleichungssystem so gut wie möglich zu lösen:

| Finde $x \in \mathbb{R}^n$ so dass das *Residuum* $r = b - Ax$ so klein wie möglich ist. Das heißt, $\|r\| = \|Ax - b\|$ minimal. |
|---|

Hierbei stellt sich die Frage, welche Norm $\|\cdot\|$ für die Minimierung von $\|r\|$ verwendet werden sollte.

**Definition 5.1.** Für $p \geq 1 \in \mathbb{N}$, $x \in \mathbb{R}^m$ ist die $p$-Norm definiert als

$$\|x\|_p = \sqrt[p]{\sum_{i=1}^{m} |x_i|^p} = (|x_1|^p + |x_2|^p + \cdots + |x_m|^p)^{1/p}.$$

Häufig verwendete Beispiele für $p$-Normen sind:

$$\|x\|_1 = \sum_{i=1}^{m} |x_i|$$

$$\|x\|_2 = \sqrt{\sum_{i=1}^{m} x_i^2}$$

$$\|x\|_\infty = \max\{|x_i| \mid i = 1, \ldots, m\}$$

Da das analytische Minimieren einer Norm das Verschwinden der ersten Ableitung erfordert, benötigen wir eine differenzierbare Norm. Von den Beispielen oben ist nur die 2-Norm differenzierbar. (Genauer: wir minimieren $\|x\|_2^2$, das ist differenzierbar.) Die Verwendung der 2-Norm hat viele Vorteile.

- Die quadrierte 2-Norm $\|r\|_2^2$ ist differenzierbar.

- Die Ableitung der quadrierten 2-Norm $\|r\|_2$ ist eine lineare Funktion, was am Ende zu einem linearen Gleichungssystem führt.
- Die 2-Norm $\|r\|_2$ erlaubt geometrische Interpretationen, da sie dem euklidischen (also in unserer Welt dem realen) Abstand entspricht und einen direkten Zusammenhang zum Skalarprodukt und daher zu Winkeln zwischen Vektoren herstellt.

Aus diesen Gründen wird im folgenden die 2-Norm für die Minimierung des Residuums verwendet. Soweit nicht anders erwähnt, sei mit der Vektor-Norm $\|\cdot\|$ ab jetzt die 2-Norm bezeichnet. Die Minimierung von $\|r\| = \|Ax-b\|$ wird *Methode der kleinsten (Fehler-)Quadrate* (im Englischen *least squares)* genannt und geht auf Gauß (1809) zurück.

## 5.1 Methode der kleinsten Quadrate

Die Kurven-Approximation lässt sich wie folgt als Least-Squares-Problem formulieren: Finde $x \in \mathbb{R}^n$ so, dass $\|b-Ax\|_2^2$ minimal wird. (Damit wird dann natürlich auch $\|Ax-b\|_2 = \sqrt{\|b-Ax\|_2^2}$ minimal.) Oder anders formuliert: Minimiere die (reellwertige!) Funktion

$$R : \mathbb{R}^n \to \mathbb{R}, \quad R(x) = \|b-Ax\|^2.$$

Recall Mathe 2: Eine notwendige Bedingung für ein Minimum ist, dass der Gradient 0 wird; dass also die ersten partiellen Ableitungen bezüglich aller $x_i$ verschwinden:

$$\frac{\partial}{\partial x_i} R(x) = 0, \quad i = 1, \ldots, n.$$

Da $R$ eine quadratische Funktion in den Variablen $x_i$ ist, gibt es genau ein Extremum, an dem die ersten partiellen Ableitungen verschwinden. Da unsere Fehlerfunktion nach oben nicht beschränkt ist, muss es sich bei dem Extremum um ein Minimum handeln. Formell bedeutet dies, dass die zweiten Ableitungen von $R$ positiv sein müssen, d.h., dass die Hesse-Matrix positiv definit sein muss.

Wir werden im Folgenden drei verschiedene Herleitungen der Lösung des Least-Squares-Problems vorstellen. Die ersten beiden finden analytisch die Nullstelle der ersten Ableitung, die dritte Methode verwendet eine geometrische Herleitung. Das resultierende lineare Gleichungssystem ist bei allen Methoden das gleiche.

**1: Herleitung mit Analysis.** Das Residuum ist

$$b-Ax = \begin{pmatrix} r_1 \\ \vdots \\ r_m \end{pmatrix} = \begin{pmatrix} b_1 - \sum_{j=1}^n a_{1j}x_j \\ \vdots \\ b_m - \sum_{j=1}^n a_{mj}x_j \end{pmatrix}.$$

Dann ist $\|R(x)\|^2 = \sum_{i=1}^m r_i^2$. Die möglichen Extrema von $R$ sind die Nullstellen des Gradienten von $R$, also des Vektors der ersten partiellen Ableitungen von $R(x)$:

$$\Big(\frac{\partial}{\partial x_1} R(x), \cdots, \frac{\partial}{\partial x_n} R(x)\Big).$$

Berechnen wir erstmal

$$\frac{\partial}{\partial x_k} r_i^2 = \frac{\partial}{\partial x_k}\Big(b_i - \sum_{j=1}^n a_{ij}x_j\Big)^2 \overset{\text{(Kettenregel)}}{=} 2\Big(b_i - \sum_{j=1}^n a_{ij}x_j\Big)\frac{\partial}{\partial x_k}(b_i - \sum_{j=1}^n a_{ij}x_j) = 2r_i \frac{\partial}{\partial x_k}(b_i - \sum_{j=1}^n a_{ij}x_j).$$

Wegen $\frac{\partial}{\partial x_k} b_i = 0$ (Konstante!) und $\frac{\partial}{\partial x_k} a_{ij} x_j = 0$ für $k \neq j$ bleibt nur übrig

$$2r_i \frac{\partial}{\partial x_k}\big(-a_{ik}x_k\big) = -2r_i \frac{\partial}{\partial x_k} a_{ik}x_k = -2a_{ik}r_i$$

Die partielle Ableitung $\frac{\partial}{\partial x_j} R(x)$ ergibt sich damit als

$$\frac{\partial}{\partial x_k} R(x) = \frac{\partial}{\partial x_k} \sum_{i=1}^{m} r_i^2 = \sum_{i=1}^{m} \frac{\partial}{\partial x_k} r_i^2 = \sum_{i=1}^{m} -2a_{ik}r_i = -2\sum_{i=1}^{m} a_{ik}\Big(b_i - \sum_{j=1}^{n} a_{ij}x_j\Big).$$

Da alle partiellen Ableitungen verschwinden müssen, erhalten wir die $n$ Bedingungen

$$\sum_{i=1}^{m} a_{ik}\Big(b_i - \sum_{j=1}^{n} a_{ij}x_j\Big) = 0, \quad \text{also}$$

$$\sum_{i=1}^{m} a_{ik} \sum_{j=1}^{n} a_{ij}x_j = \sum_{i=1}^{m} a_{ik}b_i, \quad (k = 1, \ldots, n)$$

Durch scharfes Hingucken erkennt man, dass jede Bedingung $j = 1, \ldots, n$ einer Zeile des folgenden $(n \times n)$-Gleichungssystems entspricht:

$$A^T Ax = A^T b. \tag{5.1}$$

Die Lösung $x$ dieses linearen Gleichungssystems liefert also das gesuchte Minimum.

Die Gleichungen (5.1) werden auch **Normalgleichungen** (oder Normalengleichungen) genannt (englisch *normal equations*). Sofern $A^T A$ invertierbar ist (das heißt, wenn $A$ vollen Ranf $n$ hat), lässt sich die Lösung schreiben als $x = \big(A^T A\big)^{-1} A^T b =: A^+ b$.

Die Matrix $A^+ = (A^T A)^{-1} A^T$ heißt **Pseudoinverse** von $A$. Das ist ein Notbehelf, falls eine Matrix $A$ keine Inverse hat. Es gibt verschiedene solcher Pseudoinversen, aber das hier ist eine sehr gebräuchliche. Denn sie passt ja unter Anderem prima zur Situation hier: $x = A^{-1}b$ geht nicht, das nächstbeste ist $x = A^+ b$.

**2. Herleitung mit Vektor-Analysis.** Im letzten Abschnitt haben wir eine skalare Gleichung für jede partielle Ableitung aufgestellt und diese dann wieder in Matrix-Notation zusammengefasst. Mit ein paar einfachen Ableitungsregeln für Matrizen und Vektoren lassen sich die Normalgleichungen mit deutlich weniger Aufwand herleiten. Wir führen eine kompakte Schreibweise ein: für $x = (x_1, \ldots, x_n)^T \in \mathbb{R}^n$ ist

$$\frac{\partial R(x)}{\partial x} := \Big(\frac{\partial}{\partial x_1} R(x), \cdots, \frac{\partial}{\partial x_n} R(x)\Big)^T,$$

und $\frac{\partial Ax}{\partial x}$ ist die Matrix $(\frac{\partial (Ax)_i}{\partial x_j})_{ij}$. Die Ableitungsregeln, die wir brauchen, sind:

- $\frac{\partial b^T x}{\partial x} = b, \quad \frac{\partial x^T b}{\partial x} = b$
- $\frac{\partial Ax}{\partial x} = A, \quad \frac{\partial x^T A}{\partial x} = A$
- $\frac{\partial x^T Ax}{\partial x} = (A + A^T)x$

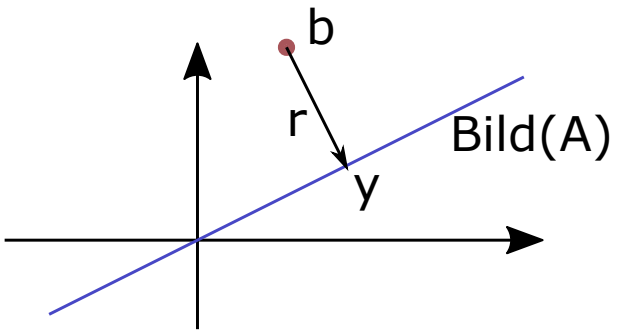



Figure 4: Das $y$, das $\|Ax-b\|$ minimiert, ist die orthogonale Projektion von $b$ auf Bild($A$).

Die kann man sich im Prinzip mit dem, was man in Mathe 2 über partielle Ableitungen gelernt hat, selber herleiten. Es ist mathematisch dasselbe, nur eine effizientere Schreibweise. Denn nun können wir z.B. den Gradient von $R(x)$ so berechnen: wir benutzen $\|r\|_2^2 = r^T r$. Damit

$$\begin{aligned} R(x) &= \|b-Ax\|_2^2 = (b-Ax)^T(b-Ax) = x^TA^TAx - x^TA^Tb - b^TAx + b^Tb \\ &= x^TA^TAx - 2x^TA^Tb + b^Tb \quad (\text{denn } x^TAb = b^TAx \in \mathbb{R}) \end{aligned}$$

(vgl binomische Formel!) Mit Hilfe der obigen Ableitungsregeln (wobei die symmetrische Matrix $A^TA = (A^TA)^T$ hier die Rolle von $A$ oben übernimmt(, können wir den Gradienten von $R$ sehr einfach berechnen als

$$\frac{\partial}{\partial x}\left(x^TA^TAx - 2x^TA^Tb + b^Tb\right) = (A^TA + (A^TA)^T)x - 2A^Tb = 2A^TAx - 2A^Tb.$$

Die Bedingung, dass das 0 wird, führt direkt zu den Normalgleichungen.

**3. Geometrische Herleitung.** Abbildung 4 liefert die Grundlage für die geometrische Herleitung der Normalengleichungen. Das LGS $Ax = b$ kann nicht exakt gelöst werden, da $b$ nicht im Bild von $A$ liegt und es daher kein $x$ gibt, für das $Ax = b$ gilt. Geometrisch ist klar (und mit Linearer Algebra recht einfach zu zeigen), dass $y$ als orthogonale Projektion von $b$ auf das Bild von $A$ die beste Approximation ist. Das heißt, dass das Residuum $r = b - Ax$ minimal wird, wenn $r$ orthogonal zum Bild von $A$ ist. Wir suchen also $x$, so dass $Ax = y$. Ausgehend von dieser Beobachtung ergeben sich die Normalengleichungen wie folgt:

Wir erinnern uns, dass das Bild von $A$ von den Spalten $a_1, \ldots, a_n$ von $A$ aufgespannt wird. Es ist also $r$ orthogonal zum Bild von $A$ genau dann, wenn $r$ orthogonal zu allen $a_i$ ist. Das heißt:

$$a_i^T r = 0 \quad \text{für } i = 1, \ldots, n \;\Leftrightarrow\; A^Tr = 0 \;\Leftrightarrow\; A^T(b-Ax) = 0 \;\Leftrightarrow\; A^TAx = A^Tb.$$

# 6 Cholesky-Zerlegung

Mit den Normalgleichungen haben wir das ursprüngliche, überbestimmte LGS in ein quadratisches LGS überführt. Die Normalgleichungen (5.1) können also z.B. mit dem uns bereits bekannten LU-Löser aus Kapitel 4 gelöst werden. (Das $A^Tb$ der Normalengleichung ist dann das $b$ bei dem LU-Löser, das lässt sich mit Aufwand $O(n^2)$ berechnen, also schneller als die LU-Zerlegung.) Wir werden im nächsten Kapitel sehen, dass das nicht optimal ist. Die LU-Zerlegung verlangt keine spezielle Struktur der Matrix. Aber unsere Matrix $A^TA$ (mit $A \in \mathbb{R}^{m\times n}$, für $m > n$) hat doch eine spezielle Struktur: 7. Mai

- $A^TA$ ist symmetrisch: $(A^TA)^T = A^TA$ (Warum genau?)
- Wenn $A$ vollen Rang (also $n$) hat, dann ist $A^TA$ invertierbar. (Dies folgt z.B. aus der Tatsache, dass es dann ein eindeutiges $x$ gibt, so dass $Ax = y$.)
- Wenn $A$ vollen Rang hat, dann ist $A^TA$ *positiv definit*, d.h.,

  $$x^T(A^TA)x > 0 \quad \text{für alle } x \neq 0.$$

  Wenn nämlich $A$ vollen Rang hat, dann gilt für $x \neq 0$ auch $Ax \neq 0$. Daher gilt $0 < \|Ax\|^2 = (Ax)^TAx = x^TA^TAx$ .

Insgesamt erhalten wir für eine Matrix $A$ mit vollem Spaltenrang eine Matrix $A^TA$ der Normalgleichungen, die symmetrisch, positiv definit und invertierbar ist.

In diesem Kapitel stellen wir daher ein Verfahren vor, dass auf symmetrisch positiv definite Matrizen abgestimmt ist: Die Cholesky-Zerlegung. Im Gegensatz zur LU-Zerlegung, die beliebige $m \times m$ Gleichungssysteme $Ax = b$ lösen kann, ist die Cholesky-Zerlegung auf symmetrisch positiv definite Matrizen spezialisiert. Die spezielle Struktur dieser Matrizen führt dann zu einem effizienteren Löser als die LU-Zerlegung. Wie oben diskutiert, sind die Normalgleichungen $A^TAx = A^Tb$ eines überbestimmten Gleichungssystems $Ax = b$ ein Beispiel eines symmetrisch positiv definiten Systems. Viele weitere in der Praxis häufig vorkommende und sehr relevante Probleme führen ebenso auf symmetrisch positiv definite Matrizen, z.B. die Minimierung quadratischer Energien aus physikalischen Simulationen, oder Kovarianzmatrizen in der Stochastik.

Die wesentliche Idee der Cholesky-Zerlegung ist wie folgt: Wenn A symmetrisch positiv definit ist, dann existiert die Cholesky-Zerlegung A = $LL^T$, wobei $L$ eine untere Dreiecksmatrix ist. Ähnlich zur LU-Zerlegung teilt sich das Lösen des Gleichungssystems $Ax = b$ in folgende drei Schritte auf:

1. Cholesky-Zerlegung: $A = LL^T$.
2. Löse $Ly = b$ durch Vorwärts-Substitution.
3. Löse $L^Tx = y$ durch Rückwärts-Substitution.

Da die Schritte 2 und 3 analog zur LU-Zerlegung erfolgen (siehe Kapitel 4), wird in diesem Kapitel nur die eigentliche Zerlegung $A = LL^T$ diskutiert.

## 6.1 Symmetrisch Positiv Definite Matrizen (SPD)

Bevor wir die Cholesky-Zerlegung darstellen, präzisieren wir zunächst die Definition einer symmetrisch positiv definiten Matrix und betrachten einige der daraus folgenden Eigenschaften.

**Definition 6.1.** Eine Matrix $A \in \mathbb{R}^{m\times m}$ heißt **symmetrisch positiv definit** (kurz: SPD), wenn $A = A^T$ (symmetrisch) und $x^TAx > 0$ für alle $x \in \mathbb{R}^m$ mit $x \neq 0$.

Übrigens heißt $A$ *positiv semidefinit*, falls oben nur $x^TAX \geq 0$ gilt, und analog *negativ definit*, falls oben $x^TAx < 0$ für $x \neq 0$, sowie *negativ semidefinit*, falls nur $x^TAx \leq 0$.

Eine symmetrische reelle Matrix $A \in \mathbb{R}^{m\times m}$ hat $m$ reelle (!) Eigenwerte und ist diagonalisierbar. Sie ist positiv definit genau dann, wenn alle ihre Eigenwerte positiv sind. (Analog: alle Eigenwerte nichtnegativ g.d.w. positiv semidefinit, alle Eigenwerte negativ g.d.w. negativ semidefinit usw.)

Folgende Eigenschaften einer symmetrisch positiv definiten Matrix $A \in \mathbb{R}^{m\times m}$ werden wir bei der Konstruktion der Cholesky-Zerlegung ausnutzen:

1. Ist $m \geq n$, so ist für eine Matrix $M \in \mathbb{R}^{m\times n}$ von vollem Rang die Matrix $M^T AM$ ebenfalls SPD.
2. Jede der Untermatrizen $A_{k..\ell,k../ell} = (a_{ij})_{k\leq i,j\leq \ell}$ von $A$ ist ebenfalls SPD.
3. Insbesondere ist jeder Eintrag auf der Hauptdiagonalen eine positive reelle Zahl.

Die Untermatrizen $A_{k,\ell}$ aus Punkt 2 heißen **Hauptminoren**. (Ein Hauptminor ist eigentlich etwas allgemeiner definiert, aber das hier sind alles welche.) Die Aussagen oben lassen sich alle fix herleiten mit etwas Linearer Algebra.

Zu 1.: Die Matrix $M^T AM$ ist symmetrisch, weil $(M^T AM)^T = M^T A^T M = M^T AM$. Sie ist positiv definit, da bei vollem Rang von $M$ für $x \neq 0$ auch $Mx \neq 0$ (hier brauchen wir $m \geq n$, sonst ist das falsch), und somit $x^T (M^T AM)x = (Mx)^T A(Mx) > 0$.

Zu 2.: Der Hauptminor $A_{k,\ell}$ lässt sich in der Form $M^T AM$ schreiben, wenn wir $M$ geeignet als $m \times n$ Matrix mit je einer 1 in jeder Spalte und Nullen überall sonst wählen. Das sieht man vielleicht am Besten an einem Beispiel:

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 6 & 7 \\ 10 & 11 \end{pmatrix}$$

Daraus folgt nach Eigenschaft 1, dass jedes solche $A_{k,\ell}$ positiv definit ist.

Zu 3.: Das Diagonalelement ist die $1 \times 1$-Matrix $A_{k,k+1} = a$. Wegen positiv definit gilt $xax = ax^2 > 0$. Eine positiv definite $1 \times 1$-Matrix ist also eine positive reelle Zahl.

## 6.2 Choleskyzerlegung theoretisch

Die formale Grundlage der Cholesky-Faktorisierung ist Satz 6.2 unten. Der Beweis ist vollständige Induktion über $m$, wobei $A \in \mathbb{R}^{m\times m}$. Dazu überlegen wir uns zunächst, wie sich Matrizen

$$A = \begin{pmatrix} a & 0^T \\ v & A' \end{pmatrix} \quad \text{mit } a \in \mathbb{R}, v \in \mathbb{R}^{m-1}, A' \in \mathbb{R}^{(m-1)\times(m-1)}$$

multiplizieren. Wir brauchen es hier so:

$$\begin{pmatrix} a & 0^T \\ v & A' \end{pmatrix} \cdot \begin{pmatrix} b & w^T \\ 0 & B' \end{pmatrix} = \begin{pmatrix} ab & aw^T \\ bv & A' \cdot B' + v \cdot w^T \end{pmatrix}$$

Das überlegt man sich sich blockweise. Ganz oben links etwa hat man

$$a \cdot b + 0 \cdot 0 + \cdots + 0 \cdot 0 = a \cdot b$$

usw. Unten rechts ist es am schwierigsten zu sehen: Klar steht da schon Mal $A' \cdot B'$, und in Zeile $i$ und Spalte $j$ kommt noch genau einmal $v_i \cdot w_j$ dazu. Also der Eintrag $m_{i,j}$ der Matrix $M = v \cdot w^T$. Damit können wir nun zeigen:

**Satz 6.2.** *$A \in \mathbb{R}^{m\times m}$ ist symmetrisch positiv definit genau dann, wenn es eine Faktorisierung $A = LL^T$ gibt, wobei L eine untere Dreiecksmatrix ist. Mit der Forderung $\ell_{ii} > 0$ ist L eindeutig.*

*Proof.* Vollständige Induktion über $m$:

Induktionsanfang: $m = 1$: Dann ist $A = (a_{11})$. Weil $A$ symmetrisch positiv definit ist, gilt ja $xa_{11}x = a_{11}x^2 > 0$ für alle $x \in \mathbb{R}$ mit $x \neq 0$. Daraus folgt $a_{11} > 0$. Dann ist $L = (\ell_{11}) = (\sqrt{a_{11}})$.

Induktionsschluss: Wir nehmen an, der Satz gilt für $m-1$. Sei $A \in \mathbb{R}^{m,m}$,

$$A = \begin{pmatrix} a_{11} & w \\ w^T & A' \end{pmatrix}$$

Wir können $A$ schreiben als

$$A = \underbrace{\begin{pmatrix} \sqrt{a_{11}} & 0 \\ \frac{1}{\sqrt{a_{11}}}w & E \end{pmatrix}}_{L_1} \cdot \underbrace{\begin{pmatrix} 1 & 0 \\ 0 & A' - \frac{1}{a_{1,1}} w \cdot w^T \end{pmatrix}}_{A_1} \cdot \underbrace{\begin{pmatrix} \sqrt{a_{11}} & \frac{1}{\sqrt{a_{11}}}w^T \\ 0 & E \end{pmatrix}}_{L_1^T},$$

mit $A' \in \mathbb{R}^{(m-1)\times(m-1)}$. Denn Ausmultiplizieren liefert

$$L_1 \cdot (A_1 \cdot L_1^T) = \begin{pmatrix} \sqrt{a_{11}} & 0 \\ \frac{1}{\sqrt{a_{11}}}w & E \end{pmatrix} \cdot \begin{pmatrix} \sqrt{a_{11}} & \frac{1}{\sqrt{a_{11}}}w^T \\ 0 & A' - \frac{1}{a_{11}} w \cdot w^T \end{pmatrix} = \begin{pmatrix} a_{1,1} & w^T \\ w & A' \end{pmatrix},$$

vergleiche die Überlegung vor diesem Satz.

Da $L_1$ vollen Rang hat, folgt wegen $A = L_1 A_1 L_1^T$ dass

$$A_1 = L_1^{-1} A (L_1^T)^{-1} = \left((L_1^{-1})^T\right)^T A (L_1^{-1})^T$$

aus Eigenschaft 1 vom Anfang dieses Abschnitts, dass $A_1$ symmetrisch positiv definit ist (mit $M = (L_1^{-1})^T$. Aus Eigenschaft 2 folgt dann, dass auch $\tilde{A} = A' - \frac{1}{a_{1,1}} w \cdot w^T$ als Untermatrix von $A_1$ symmetrisch positiv definit ist. Nach Induktionsannahme gilt dann $\tilde{A} = \tilde{L}\tilde{L}^T$ für eine Dreicksmatrix $\tilde{L}$. Daraus folgt:

$$A = \begin{pmatrix} \sqrt{a_{11}} & 0 \\ \frac{w}{\sqrt{a_{11}}} & E \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ 0 & \tilde{L}\tilde{L}^T \end{pmatrix} \cdot \begin{pmatrix} \sqrt{a_{11}} & \frac{w^T}{\sqrt{a_{11}}} \\ 0 & E \end{pmatrix} = \begin{pmatrix} \sqrt{a_{11}} & 0 \\ \frac{w}{\sqrt{a_{11}}} & \tilde{L} \end{pmatrix} \cdot \begin{pmatrix} \sqrt{a_{11}} & \frac{w^T}{\sqrt{a_{11}}} \\ 0 & \tilde{L}^T \end{pmatrix}$$

□

Hier sieht man auch, dass die Idee eine symmetrische Variante der LU-Zerlegung ist: wir multiplizieren nun von links *und* rechts mit $L_i$ (bzw $L_i^T$), bis in der Mitte nur noch die Einheitsmatrix steht. Also...

## 6.3 Choleskyzerlegung praktisch

... berechnet sich die Choleskyzerlegung in der Praxis sehr einfach: Wir stellen Zeilenstufenform her, wobei wir immer $\ell_{kk} = 1$ benutzen, zum Beispiel für $A = \left(\begin{smallmatrix} 4 & 2 & 2 \\ 2 & 2 & 2 \\ 2 & 2 & 11 \end{smallmatrix}\right)$:

$$\begin{pmatrix} 1 & 0 & 0 \\ -\frac{1}{2} & 1 & 0 \\ -\frac{1}{2} & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 4 & 2 & 2 \\ 2 & 2 & 2 \\ 2 & 2 & 11 \end{pmatrix} = \begin{pmatrix} 4 & 2 & 2 \\ 0 & 1 & 1 \\ 0 & 1 & 10 \end{pmatrix}, \quad \text{also}$$

$$\begin{pmatrix} 4 & 2 & 2 \\ 2 & 2 & 2 \\ 2 & 2 & 11 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{2} & 1 & 0 \\ \frac{1}{2} & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 4 & 2 & 2 \\ 0 & 1 & 1 \\ 0 & 1 & 10 \end{pmatrix}, \quad \text{und}$$

$$\begin{pmatrix} 4 & 2 & 2 \\ 2 & 2 & 2 \\ 2 & 2 & 11 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{2} & 1 & 0 \\ \frac{1}{2} & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 4 & 2 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 9 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{2} & 1 & 0 \\ \frac{1}{2} & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 4 & 2 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 9 \end{pmatrix}.$$

Das ist nicht ganz korrekt (eigentlich müssten wir immer $\ell_{kk} = \frac{1}{\sqrt{a_{kk}}}$ benutzen), aber leicht zu reparieren: wir nehmen in der rechten Matrix der $k$-ten Zeile ein $\sqrt{a_{kk}}$ weg und spendieren das der $k$-ten Spalte der linken Matrix (wegnehmen heißt: durch $\sqrt{a_{kk}}$ teilen, und spendieren heißt: mal $\sqrt{a_{kk}}$.)

$$\begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{2} & 1 & 0 \\ \frac{1}{2} & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 4 & 2 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 9 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{2} & 1 & 0 \\ \frac{1}{2} & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 3 \end{pmatrix} \cdot \begin{pmatrix} 2 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 3 \end{pmatrix} = \cdot \begin{pmatrix} 2 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 3 \end{pmatrix} \cdot \begin{pmatrix} 2 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 3 \end{pmatrix}.$$

Der Pseudo-Code für die Cholesky-Faktorisierung ist in Algorithmus 2 wiedergegeben. Er läuft nach folgendem Schema ab: $A = L_1 A_1 L_1^T$, $A_1 = L_2 A_2 L_2^T, \ldots$. Dieser Prozess wird rekursiv wiederholt, bis sich ergibt:

$$A = L_1 \cdots L_m \cdot E \cdot L_m^T \cdots L_1^T.$$

Das Produkt der unteren Dreiecksmatrizen $L_i$ kann dabei analog zur LU-Zerlegung einfach bestimmt werden.

Vorsicht: $\ell_{kk}$ wird in der drittletzten Zeile überschrieben, daher sollte $\frac{1}{\sqrt{\ell_{kk}}}$ vorher berechnet und gespeichert werden.

**Algorithm 2 Cholesky-Zerlegung**. Eingabe: Matrix $A$. Ausgabe: Matrix $L$ mit $A = LL^T$.

```
L = A, setze oberes Dreieck von L auf 0.
for k = 1...m do
  for j = k+1...m do
    for i = m...j do
      ℓ[i,j] = ℓ[i,j] − ℓ[i,k] ℓ[j,k]/ℓ[k,k]
    end for
  end for
  for i = m...k do
    ℓ[i,k] = ℓ[i,k]/√ℓ[k,k]
  end for
end for
```

## 6.4 Rechenaufwand und Stabilität

Die Berechnungen für die Cholesky-Faktorisierung werden von den Operationen in der inneren Schleife dominiert. Die Ausführung von $\ell[i,j] = \ell[i,j] - \ell[i,k]\frac{\ell[j,k]}{\ell[k,k]}$ ($i$ und $j$ sind die Schleifenindizes) für $i$ von $m$ bis $j$ benötigt eine Division, $m-j+1$ Multiplikationen und $m-j+1$ Subtraktionen, insgesamt also etwa $2(m-j)$ FLOP. Diese Berechnung wird für jedes $j$ von $k+1$ bis $m$ wiederholt. Die äußere

Schleife wird für jedes $k$ von 1 bis $m$ wiederholt. In der Summe ergibt sich daraus

$$\sum_{k=1}^{m}\sum_{j=k+1}^{m} 2(m-j) = 2\sum_{k=1}^{m}\sum_{j=0}^{m-k-1} j = 2\sum_{k=1}^{m}\frac{1}{2}\big((m-k)^2-(m-k)\big) = \sum_{i=0}^{m-1}(i^2-i)$$
$$= \frac{1}{6}(m-1)m(2m-2) \;-\; \frac{1}{2}(m-1)m \approx \frac{1}{3}m^3 \text{ FLOP.}$$

Die Cholesky-Faktorisierung benötigt also nur halb so viele Operationen wie die LU-Faktorisierung. Gleiches gilt aufgrund der Symmetrie der Matrix auch für den Speicherverbrauch.

Zur Stabilität: Prinzipiell gibt es zwei Stellen, an denen der Algorithmus fehlschlagen könnte:

1. Die Berechnung von $\frac{\ell_{jk}}{\ell_{kk}}$ könnte fehlschlagen, wenn $\ell_{kk} < \varepsilon_m$.
2. Die Berechnung von $\sqrt{\ell_{kk}}$ könnte fehlschlagen, wenn $\ell_{kk} < 0$.

Da aber $A$ symmetrisch positiv definit ist (und damit auch die $A_i$), folgt, dass 1. nach der Zusatzaufgabe von Blatt 3 die betragsgrößten Elemente von $A_i$ auf der Hauptdiagonalen liegen (das sind ja die $\ell_{kk}$), und 2. die Diagonalelemente $\ell_{kk}$ alle positiv sind. Somit ist für die Cholesky-Faktorisierung auch gar keine Pivotisierung nötig.

Um zu verstehen, warum auch die Cholesky-Zerlegung nicht perfekt ist (denn sie ist numerisch nicht stabiler), brauchen wir den Begriff der Konditionszahl.

# 7 Konditionszahlen

## 7.1 Kondition von allgemeinen Problemen

14. Mai

Das zu untersuchende Problem sei abstrakt als eine Funktion $f : X \to Y$ modelliert, welche bei Eingabe $y$ die Ausgabe $f(y)$ liefert. Das Problem gilt als gut konditioniert, wenn eine kleine Änderung in y auch nur eine kleine Änderung in $f(y)$ bewirkt. Umgekehrt ist ein Problem schlecht konditioniert, wenn eine kleine Änderung in $y$ eine deutlich größere Änderung in $f(y)$ zur Folge haben kann. Ist $y$ das korrekte $y$, und $\tilde{y}$ der abweichende $y$-Wert, dann suchen wir also die Zahl $\kappa_a \geq 0$, so dass

$$\|f(y)-f(\tilde{y})\| \leq \kappa_a \|y-\tilde{y}\|, \quad \text{also } \frac{\|f(y)-f(\tilde{y})\|}{\|y-\tilde{y}\|} \leq \kappa_a.$$

Wir denken uns das $\tilde{y}$ nah an $y$. Falls das $f$ differenzierbar ist, erhalten wir im Grenzwert $\kappa_a = \lim_{\tilde{y}\to y} \frac{\|f(y)-f(\tilde{y})\|}{\|y-\tilde{y}\|} = f'(y)$. Das ist schön, denn das liefert eine einfache Formel für die Kondition. Aber damit messen wir ja nur den absoluten Fehler. Wenn $f(y)$ astronomisch hoch ist, dann liefert ja eine Abweichung von 0,000001% ja immer noch ein sehr hohes $\kappa_a$. Die relevante Zahl ist eher die, die wir für den relativen Fehler erhalten:

$$\frac{\|f(\tilde{y})-f(y)\|}{\|f(y)\|} \leq \kappa \frac{\|\tilde{y}-y\|}{\|y\|}$$

(Abweichung geteilt durch $\|f(y)\|$ bzw $\|y\|$). Anschaulich soll also die Kondition nicht von der Längeneinheit, in der wir messen, abhängen: ob wir in Millimeter messen oder Kilometer soll nichts

ausmachen. Umstellen nach $\kappa$ und substituieren $\delta y := \tilde{y} - y$ liefert

$$\frac{\|f(y+\delta y) - f(y)\|}{\|\delta y\|} \cdot \frac{\|y\|}{\|f(y)\|} \leq \kappa.$$

Ist $f$ wieder differenzierbar, erhalten wir also als **Konditionszahl**

$$\kappa = \lim_{\delta \to 0} \frac{\|f(y+\delta y) - f(y)\|}{\|\delta y\|} \cdot \frac{\|y\|}{\|f(y)\|} = \frac{\|Df(y)\|\|y\|}{\|f(y)\|}.$$

Das $D$ heißt Ableitung von $f$. Für $f : \mathbb{R} \to \mathbb{R}$ ist also $Df$ einfach $f'$. Für $f : \mathbb{R}^m \to \mathbb{R}^m$ ist $Df$ die Jacobi-Matrix $J_f = \frac{\partial f}{\partial y} = \left(\frac{\partial f_i}{\partial y_j}\right)_{1 \leq i,j \leq m}$

Ein Problem gilt nun als gut konditioniert, wenn $\kappa$ klein ist (Faustregel: $10^0$ bis $10^2$). Ein schlecht konditioniertes Problem liegt vor, wenn $\kappa$ groß ist (Faustregel: ab $10^5$ oder besser, $1/\kappa$ kommt in die Nähe der Rechengenauigkeit).

Wir wollen ja eigentlich die Kondition von Matrix-Vektor-Multiplikationen betrachten. Dafür wählen wir als Problemfunktion $f(y) = Ay$ (Denn die Eingabe $y$ ist hier ja $b$, Ausgabe ist $A^{-1}b$). Da die Ableitung $Df$ bzw. die Jacobi-Matrix $J_f$ die Matrix $A$ selbst ist (siehe Kapitel 5.1, "Herleitung mit Vektoranalysis") gilt für die Kondition dieses Problems

$$\kappa = \frac{\|A\|\|x\|}{\|Ax\|}.$$

Wenn wir das ansehen, sind die Normen $\|x\|$ und $\|Ax\|$ ja bereits bekannt: wir können uns eine $p$-Norm aussuchen, etwa die 2-Norm. Allerdings müssen wir ja noch klären, was die Norm $\|A\|$ einer Matrix $A$ sein soll.

## 7.2 Matrixnormen

Für eine Matrix-Norm $\|\cdot\|$ auf $\mathbb{R}^{m \times n}$ muss (wie für jede Norm) gelten:

1. $\|A\| \geq 0$, und $\|A\| = 0 \Leftrightarrow A = 0$,
2. $\|\alpha A\| = |\alpha| \cdot \|A\|$, und
3. $\|A + B\| \leq \|A\| + \|B\|$.

Ein sehr häufig verwendete Matrix-Norm ist die Frobenius-Norm, welche analog zur 2-Norm von Vektoren die Wurzel der Summe der quadrierten Elemente berechnet:

$$\|A\|_F := \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}^2}$$

Ziel bei alldem ist ja, die Fehler abzuschätzen. Damit die Fehlerschätzungen Sinn ergeben, müssen sich Vektornorm und Matrixnorm "vertragen". Eine Matrixnorm für $A \in \mathbb{R}^{m \times m}$ heißt daher **verträglich** mit einer Vektornorm, falls

$$\|Ax\| \leq \|A\| \cdot \|x\| \quad \text{für alle } x \in \mathbb{R}^n,\ A \in \mathbb{R}^{m \times m}.$$

Daher werden oft **induzierte** Matrix-Normen verwendet, welche über Vektor-Normen in Definitions- und Bildbereich von $A$ definiert sind; und zwar gerade so, *dass* sie verträglich sind.

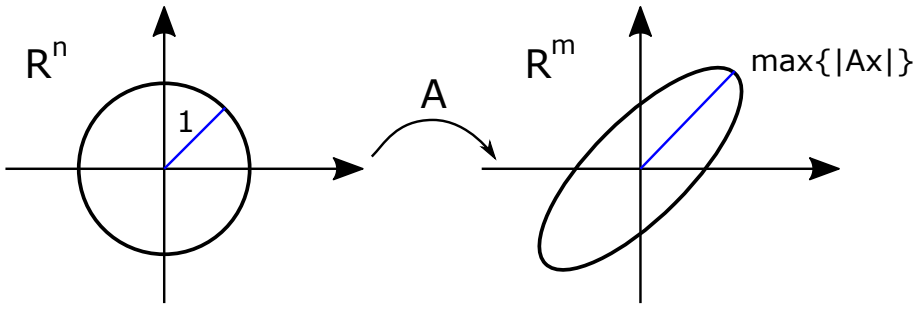



Figure 5: Die von der 2-Norm (für Vektoren) induzierte Matrixnorm von $A$ ist die maximale Streckung des Einheitskreises durch $A$.

**Definition 7.1.** Sei $A \in \mathbb{R}^{m\times n}$ und seien $\|\cdot\|_{V_1}$ und $\|\cdot\|_{V_2}$ Vektor-Normen auf $\mathbb{R}^m$ und $\mathbb{R}^n$. Dann ist die durch $\|\cdot\|_{V_1}$ und $\|\cdot\|_{V_2}$ induzierte Matrix-Norm

$$\|A\| = \max_{x\neq 0} \frac{\|Ax\|_{V_1}}{\|x\|_{V_2}} = \max_{\|x\|_{V_2}=1} \|Ax\|_{V_1}.$$

Für quadratische Matrizen ist die induzierte Matrixnorm automatisch verträglich mit der Vektornorm. (Für nichtquadratische Matrizen auch irgendwie, nur ist das aufwändiger zu definieren, da man zwei Vektornormen berücksichtigen muss.)

Die zweite Gleichung oben gilt, da Skalare (hier $\|x\|$) aus Normen herausgezogen werden können (siehe zweite Eigenschaft von Matrix-Normen). Damit ist

$$\frac{\|Ax\|}{\|x\|} = \frac{\|x\| \cdot \left\|A\frac{x}{\|x\|}\right\|}{\|x\|} = \left\|A\frac{x}{\|x\|}\right\|,$$

wobei $0 \neq x \in \mathbb{R}$, also durchläuft $\frac{x}{\|x\|}$ alle Vektoren der Länge 1. Die von der 2-Norm induzierte Matrixnorm heißt **Spektralnorm**.

**Satz 7.2.** *Ist A diagonalisierbar (also z.B. falls A symmetrisch), dann ist die Spektralnorm der Betrag $|\lambda|$ des betragsgrößten Eigenwerts $\lambda$ von A. Ansonsten ist sie der betragsgrößte **Singulärwert** von A, das heißt, die Quadratwurzel $\sqrt{|\lambda|}$ des Betrags des betragsgrößten Eigenwerts $\lambda$ von $A^TA$.*

Wir bezeichnen auch die Spektralnorm mit $\|\cdot\|_2$. Für diese kann man sich die geometrische Bedeutung sehr schön klar machen: sie misst die maximale Streckung des Einheitskreises, siehe Abbildung 5. Die Spektralnorm ist eben deshalb aufwändig zu berechnen, hat aber andere Vorteile (siehe unten).

Die von den anderen uns vertrauten $p$-Normen induzierten Matrixnormen sind:

- für $\|\cdot\|_\infty$ ergibt sich $\|A\|_\infty = \max_{i=1,\ldots,m} \sum_{j=1}^n |a_{ij}|$ (Zeilensummennorm).
- für $\|\cdot\|_1$ ergibt sich $\|A\|_1 = \max_{j=1,\ldots,n} \sum_{i=1}^m |a_{ij}|$. (Spaltensummennorm).

Nützliche Eigenschaften all dieser Matrix-Normen sind:

- $\|Ax\| \leq \|A\|\|x\|$ (**verträglich**, klar, wegen induziert),

- $\|AB\| \leq \|A\| \cdot \|B\|$ (**submultiplikativ**).

Nur für $\|\cdot\|_2$ und $\|\cdot\|_F$ gilt außerdem:

- $\|QA\| = \|A\|$ mit $Q$ orthogonal.
- $\|A\| = \|A^T\|$.

Nur für $\|\cdot\|_2$ gilt für orthogonale Matrizen $Q$ sogar $\|Q\|_2 = 1$. Punkt 4 wird für $\|\cdot\|_1$ bzw $\|\cdot\|_\infty$ zu $\|A^T\|_1 = \|A\|_\infty$ und umgekehrt. (Das sieht man direkt an den Formeln oben).

Die Bedeutung der Frobeniusmnorm liegt darin, dass man damit die (schwerer zu berechnende) Spektralnorm abschätzen kann:

$$\|A\|_2 \leq \|A\|_F.$$

Der Zusammenhang zwischen der Spektralnorm und $\|\cdot\|_1$ bzw $\|\cdot\|_\infty$ ist nicht so einfach. Es gilt

$$\frac{1}{\sqrt{m}}\|A\|_1 \leq \|A\|_2 \leq \sqrt{n}\|A\|_1 \text{ und } \frac{1}{\sqrt{n}}\|A\|_\infty \leq \|A\|_2 \leq \sqrt{m}\|A\|_\infty.$$

Außerdem gilt

$$\|A\|_2 \leq \sqrt{\|A\|_1 \cdot \|A\|_\infty}.$$

## 7.3 Konditionszahl von Matrizen

Führen wir die Überlegung von oben fort: wir wollen die Kondition des Problems "löse Ax=b" messen durch $\kappa = \frac{\|A\|\|x\|}{\|Ax\|}$. Wenn wir weiterhin annehmen, dass die Matrix $A$ invertierbar ist, dann gilt

$$\|x\| = \|A^{-1}Ax\| \leq \|A^{-1}\|\|Ax\|, \quad \text{also } \frac{\|x\|}{\|Ax\|} \leq \|A^{-1}\|.$$

Also ist $\kappa = \frac{\|A\|\|x\|}{\|Ax\|} \leq \|A\|\|A^{-1}\|$. Betrachten wir nun diese obere Schranke $\|A\|\|A^{-1}\|$ als Maß für die Kondition des Problems $f(x) = Ax$, dann hat das offenbar den Nachteil, das wir die Kondition des Problems nicht mehr so genau messen. Dafür hat das aber mehrere Vorteile:

1. Die Kondition ist nicht mehr vom Punkt $x$ abhängig.
2. Es macht keinen Unterschied, ob man die Multiplikation mit $A$ oder $A^{-1}$ untersucht. Daher hat das Problem $f(b) = A^{-1}b$, was ja auch die Lösung $x$ liefert, die gleiche Kondition.

Daher ist $\|A\|\|A^{-1}\|$ eine nützliche Zahl und bekommt einen Namen.

**Definition 7.3.** Die **Konditionszahl** einer invertierbaren Matrix $A$ ist definiert als $\kappa(A) = \|A\|\|A^{-1}\|$. Für nicht invertierbare Matrizen A ist die Konditionszahl mittels der Pseudoinversen $A^+$ definiert als $\kappa(A) = \|A\|\|A^+\|$.

Zur Pseudoinversen $A^+$ siehe Kapitel 5.1.

Die Konditionszahl hängt nun immer noch von der gewählten Matrixnorm ab. Die 2-Norm wird oft benutzt, da sie die geometrische Interpretation in Abbildung 5 hat: die 2-Norm ist einfach der größte

Streckungsfaktor. Das ist dann für diagonalisierbare Matrizen einfach der größte Eigenwert. Für die Konditionszahl ergibt sich also "größter Streckungsfaktor von $A$ mal größter Streckungs-Faktor von $A^{-1}$". Der größte Eigenwert von $A^{-1}$ ist einfach der Kehrwert des kleinsten von $A$. Das heißt, falls $A$ diagonalisierbar: $\kappa(A)$ gleich größter Eigenwert von $A$ geteilt durch kleinster Eigenwert von $A$. (Sonst entsprechend über Singulärwerte.) Außerdem verhält sich die 2-Norm brav bezüglich orthogonaler Matrizen (siehe oben).

Insgesamt erhalten wir nun folgende Resultate für die Fehlerabschätzung (der Einfachheit halber nur für quadratische Matrizen, das geht auch allgemeiner, aber schon die Formulierung wird dann sehr technisch). Die Idee ist, die Abweichung $\Delta x$ von der korrekten Lösung $x$ zu $Ax = b$ messen in Abhängigkeit von $\Delta b$, der Abweichung von $b$ (z.B. dem Fehler in einem gerundeten $b$), bzw in Abhängigkeit von $\Delta A$, der Abweichung von $A$ (also z.B. dem Fehler in einem gerundeten $A$). Hier ist aber die *Eingabe* $b$, bzw $(A,b)$ (nicht $x$), und die Ausgabe das $x$. Also nennen wir der Konsistenz halber die Ausgabe $v$ (statt $x$).

**Satz 7.4.** *Seien $A, \Delta A \in \mathbb{R}^{m\times m}$, $A$ sei invertierbar und $b, \Delta b, v, \Delta v \in \mathbb{R}^m$. Mit $\|\cdot\|$ bezeichnen wir sowohl eine Vektornorm als auch die induzierte Matrixnorm.*
***Fehlereinflüsse in** $b$: Es gelte*

$$Av = b \quad \textit{und} \quad A(v+\Delta v) = (b+\Delta b).$$

*Dann gilt für den absoluten und den relativen Fehler*

$$\|\Delta v\| \le \|A^{-1}\| \cdot \|\Delta b\|, \qquad \frac{\|\Delta v\|}{\|v\|} \le \kappa(A)\frac{\|\Delta b\|}{\|b\|}.$$

***Fehlereinflüsse in** $A$ **und** $b$: Es gelte $\|\Delta A\| \le \|A^{-1}\|^{-1}$ sowie*

$$Av = b \quad \textit{und} \quad (A+\Delta A)(v+\Delta v) = (b+\Delta b).$$

*Dann gilt für den relativen Fehler*

$$\frac{\|\Delta v\|}{\|v\|} \le \frac{\kappa(A)}{1-\kappa(A)\frac{\|\Delta A\|}{\|A\|}}\Big(\frac{\|\Delta A\|}{\|A\|} + \frac{\|\Delta b\|}{\|b\|}\Big).$$

*Proof.* Wir zeigen nur den ersten Teil. Wegen $Av = b$ und $A(v+\Delta v) = (b+\Delta b)$ ist $A\Delta v = \Delta b$, also $\Delta v = A^{-1}\Delta b$, also $\|\Delta v\| = \|A^{-1}\Delta b\|$. Wegen der Verträglichkeit der Normen folgt $\|\Delta v\| \le \|A^{-1}\|\|\Delta b\|$. Damit erhalten wir auch direkt

$$\frac{\|\Delta v\|}{\|v\|} \le \|A^{-1}\|\frac{\|\Delta b\|}{\|v\|} = \|A^{-1}\|\frac{\|\Delta b\|\cdot\|b\|}{\|b\|\|v\|} \overset{Av=b}{=} \|A^{-1}\|\frac{\|\Delta b\|}{\|b\|}\frac{\|Av\|}{\|v\|} \le \|A^{-1}\|\frac{\|\Delta b\|}{\|b\|}\frac{\|A\|\|v\|}{\|v\|} = \kappa(A)\frac{\|\Delta b\|}{\|b\|}.$$

□

Neben diesen exakten technischen Resultaten hilft Aufgabe 1 von Blatt 5 zu sehen, wann eine Matrix $A$ schlecht konditioniert ist, also $\kappa(A)$ groß ist: nämlich wenn die Spalten "fast" linear abhängig sind. Die Matrix $A$ in Abschnitt 4.2 hat eine besonders hohe Konditionszahl, wenn $\varepsilon$ sehr nah an 1 ist. (Die Probleme für $\varepsilon$ nah in 0 in Abschnitt 4.2 liegen an den Rundungsfehlern bei der Gleitkommaarithmetik; das misst die Konditionszahl nicht.)

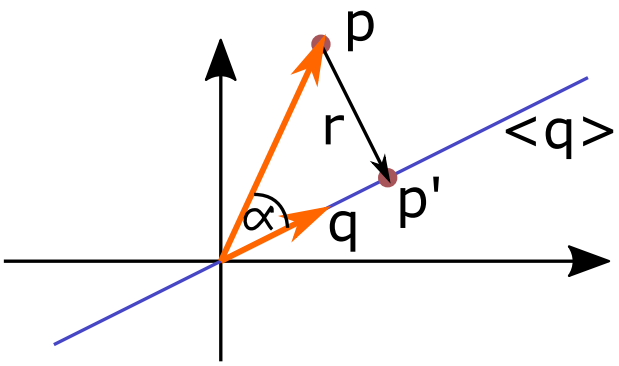



Figure 6: Orthogonale Projektion von $p$ auf den Spann $\langle q \rangle$ von $q$.

Damit erklärt sich nun auch, warum die Choleskyzerlegung nicht die beste Methode ist, um die Normalengleichung (5.1) zu lösen: Das Lösen von $A^T Ax = b$ hat die Konditionszahl $\kappa(A^T A) = \kappa(A)^2$. Falls schon $\kappa(A)$ groß ist, wird die Fehleranfälligkeit bei Cholesky nochmal quadriert. Aber die Konditionszahl gibt uns auch eine Idee, wie wir ein stabileres Verfahren finden: wegen $\|QB\|_2 = \|B\|_2$ für $Q$ orthogonal könnten wir versuchen, das $A$ als Produkt $QR$ einer orthogonalen Matrix $Q$ und einer Dreiecksmatrix $R$ zu schreiben.

# 8 Orthogonale Zerlegungen (QR-Zerlegung, SVD)

21. Mai

In Abschnitt 5.1 hatten wir uns die Normalengleichung (siehe Abbildung 4 auf Seite 23) hergeleitet mit der Überlegung, dass $\|r\| = \|Ax - b\|$ minimal wird, wenn $r$ orthogonal zu Bild$(A)$ ist. Also suchten wir das $x$, das das tut. Das war gerade die Lösung der Normalengleichung $A^T Ax = A^T b$.

Nun ist unser Ansatz etwas anders: wir projizieren das $b$ orthogonal auf Bild$(A)$. Nennen wir das Ergebnis davon $y$. *Dann* lösen wir anschließend $Ax = y$. Das geht, denn nun liegt ja $y$ im Bild von $A$. Daher müssen wir uns nun mit orthogonalen Projektionen beschäftigen und uns an orthogonale Matrizen und Vektoren erinnern.

## 8.1 Orthogonale Projektionen

Fangen wir klein an: wie projizieren wir einen Vektor $p \in \mathbb{R}^m$ auf den Spann $\langle q \rangle$ eines Vektors $q \in \mathbb{R}^m$ der Länge 1? Der Spann des Vektors $q$ ist gerade das Bild der $m \times 1$-Matrix $q$. Das Ergebnis $p'$ ist ja von der Form $c \cdot q$ mit $c \in \mathbb{R}$. Was ist das richtige $c$? Wegen Dreiecksgeometrie ist $c = \|p\| \cos(\alpha)$. Wegen Vektorgeometrie (Kapitel 2) ist

$$\cos(\alpha) = \frac{q^T p}{\|q\| \|p\|} = \frac{q^T p}{\|p\|}, \quad \text{also } \|p\| \cos(\alpha) = q^T p.$$

Damit ist

$$p' = q \cdot (\|p\| \cos(\alpha)) = q \cdot (q^T \cdot p) = (q \cdot q^T) \cdot p.$$

Die orthogonale Projektion auf $\langle q \rangle$ wird also bewirkt von der Matrix $qq^T$.

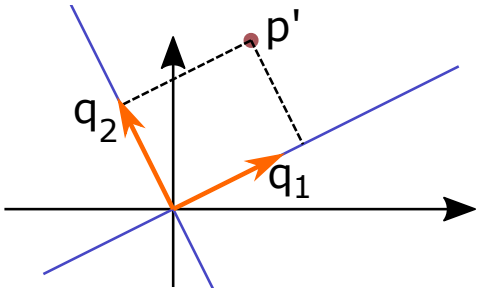



Figure 7: Orthogonale Projektion von $p$ auf den Spann $\langle q_1, q_2 \rangle$. Dabei denken wir uns $q_1$ und $q_2$ in der Bildebene und den Punkt $p$ selbst als irgendwo darüber.

Nebenbei ist $r = p - p' = (E - qq^T)p$, also bewirkt $E - qq^T$ die orthogonale Projektion auf das orthogonale Komplement $q^\perp$ von $q$.

Betrachten wir nun die orthogonale Projektion von $p$ auf den von den orthonormalen Vektoren $q_1, \ldots, q_k$ aufgespannten Unterraum $\langle q_1, \ldots, q_k \rangle$. Der projizierte Punkt $p'$ ist die Summe der einzelnen Projektionen auf die $\langle q_i \rangle$, siehe Abbildung 7. Also ist die orthogonale Projektion von $p$ auf $\langle q_1, \ldots, q_k \rangle$ gleich

$$p' = \sum_{i=1}^{k} q_i q_i^T p = \Big( \sum_{i=1}^{k} q_i q_i^T \Big) p = Q_k Q_k^T p, \qquad \text{wobei } Q_k = (q_1 | q_2 | \cdots | q_k).$$

Für das letzte = muss man sich überlegen, was denn wohl etwa

$$(q_1 | q_2) \cdot \begin{pmatrix} q_1^T \\ - \\ q_2^T \end{pmatrix}$$

ist: nämlich $(q_1^{(i)} q_1^{(j)} + q_2^{(i)} q_2^{(j)})_{i,j} = q_1 q_1^T + q_2 q_2^T$. (Zu lesen als: $q_1 = (q_1^{(1)}, q_1^{(2)}, \ldots, q_1^{(m)})^T$.)

Also tut $Q_k Q_k^T$ das, was es soll: die orthogonale Projektion auf $\langle q_1, \ldots, q_k \rangle$. Die Matrix heißt daher auch **orthogonaler Projektor** auf $\langle q_1, \ldots, q_k \rangle$.

Analog wie oben ist $E - Q_k Q_k^T$ die othogonale Projektion auf das orthogonale Komplement von $\langle q_1, \ldots, q_k \rangle$.

Man kann sich auch überlegen, was für $k = m$ passiert: dann bilden die $q_1, \ldots, q_m$ ja eine Basis des $\mathbb{R}^m$, ihr Spann ist also ganz $\mathbb{R}^m$. "Projizieren" auf $\mathbb{R}^m$ heißt dann ja einfach nichts tun. Also ist $p = p' = Q_m Q_m^T p$. Es folgt, das $Q_m Q_m^T = E$, also (wieder), dass $Q_m$ eine orthogonale Matrix ist.

## 8.2 QR-Zerlegung

Für ein überbestimmtes Gleichungssystem $Ax = b$ wollen wir also nun $b$ orthogonal auf das Bild von $A$ projizieren (siehe Abbildung 4 auf Seite 23). Diese Projektion ist schwierig zu berechnen, wenn die Spalten $a_j$ nicht orthogonal sind: $y = (A^T A)^{-1} A^T b$ (vergleiche Pseudo-Inverse in Abschnitt 5.1; insbesondere müssten wir eine im Allgemeinen aufwändige Inverse berechnen).

Die Idee ist daher, zunächst eine orthonormale Basis $\{q_1, \ldots, q_n\}$ von Bild($A$) zu konstruieren, und dann die Projektion wie im letzten Kapitel als $y = Q_n Q_n^T b$ darzustellen. (Wegen $n < m$ ist $Q_n$ *keine*

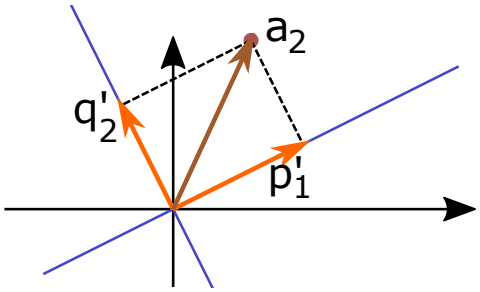



Figure 8: Der gesuchte Vektor $q_2$ entsteht durch Normieren von $q_2'$, wobei $q_2' = a_2 - p_1' = a_2 - q_1 q_1^T a_2$.

orthogonale Matrix.) Dazu lernt man in Mathe 2 das *Gram-Schmidt-Verfahren*. Das kann man jetzt vielleicht besser verstehen:

Das Bild von $A$ ist der spann der Spaltenvektoren von $A$. Sei also $a_i$ die $i$-te Spalte von $A$.

Die Konstruktion der Basisvektoren $q_i$ aus den $a_i$ erfolgt Schritt für Schritt:

- Finde $q_1'$ , so dass $\langle a_1 \rangle = \langle q_1 \rangle$; das $q_1$ ergibt sich durch normieren: $q_1 = q_1'/\|q_1'\|$.
- Finde $q_2'$, so dass $\langle a_1, a_2 \rangle = \langle q_1, q_2' \rangle$; dann normieren: $q_2 = q_2'/\|q_2'\|$; und allgemein
- Finde $q_k'$ , so dass $\langle a_1, \ldots, a_k \rangle = \langle q_1, \ldots, q_{k-1}, q_k' \rangle$; dann normieren: $q_k = q_k'/\|q_k'\|$

Mittels Gram-Schmidt wird das nach dieser Idee berechnet: wenn $p_1' = (q_1 q_1^T) a_2$ die orthogonale Projektion von $a_2$ auf $\langle q_1 \rangle$ ist, ist ja $q_2'$ das was übrigbleibt:

$$q_2' = a_2 - p_1' = a_2 - (q_1 q_1^T) a_2$$

(siehe Abbildung 8). Offenbar ist dann auch $\langle a_1, a_2 \rangle = \langle q_1', q_2' \rangle$. Damit ist $q_2'$ orthogonal zu $\langle q_1 \rangle$, muss aber noch normiert werden: $q_2 = q_2'/\|q_2'\|$. Analog ist $q_3'$ das, was von $a_3$ übrigbleibt, wenn man ihm die $q_1$- und $q_2$-Anteile wegnimmt. Das führt zu folgendem Verfahren:

**Gram-Schmidtsches Orthogonalisierungsverfahren:** Eingabe: linear unabhängige Vektoren $a_1, \ldots, a_n \in \mathbb{R}^m$

$$\begin{aligned}
q_1' &= a_1; & q_1 &= \tfrac{q_1'}{\|q_1'\|} \\
q_2' &= a_2 - q_1 q_1^T a_2; & q_2 &= \tfrac{q_2'}{\|q_2'\|} \\
q_3' &= a_3 - q_1 q_1^T a_3 - q_2 q_2^T a_3; & q_3 &= \tfrac{q_3'}{\|q_3'\|} \\
&\vdots & &\vdots \\
q_k' &= a_k - q_1 q_1^T a_k - q_2 q_2^T a_k - \cdots q_{k-1} q_{k-1}^T a_k & q_k &= \tfrac{q_k'}{\|q_k'\|}
\end{aligned}$$

Man macht solange weiter, bis $k = n$ ist, dann hat man eine ON-Basis $q_1, \ldots q_n$ gefunden. Oder solange bis ein $q_j = 0$ ist, dann war der Input $a_1, \ldots, a_n$ linear abhängig (also keine Basis).

Man beachte: das $q_1 q_1^T a_2$ können wir lesen als $(q_1 q_1^T) a_2$ (Matrix mal Vektor) oder als $q_1 (q_1^T a_2)$ (Vektor mal Zahl), je nachdem, was gerade günstiger ist.

Das liefert die Matrix $Q$. Die Matrix $R$ bekommen wir so: Der Vektor $q_j$ ist ja darstellbar in der Basis $a_1, \ldots, a_j$. Umgekehrt ist der Vektor $a_j$ darstellbar in der Basis $q_1, \ldots, q_j$, für $j = 1, \ldots, n$. Im zweiten Fall ergibt sich:

$$\begin{aligned} a_1 &= r_{11} q_1 \\ a_2 &= r_{12} q_1 + r_{22} q_2 \\ &\vdots \\ a_n &= r_{1n} q_1 + r_{2n} q_2 + \cdots + r_{nn} q_n \end{aligned}$$

Da steht eine Matrixgleichung, und rechts steht das Produkt einer (oberen) Dreicksmatrix $R$ mit einer orthogonalen Matrix $Q = (q_1, \ldots, q_n)$. Wir kennen nun die $a_i$ und $q_i$, wir brauchen noch die $r_{ij}$. Vergleichen wir diese Formeln mit den Formeln im Gram-Schmidt-Verfahren, sieht man doch

$$\begin{aligned} a_1 &= \|q_1'\| q_1 \\ a_2 &= (q_1^T a_2) q_1 + \|q_2'\| q_2 \\ &\vdots \\ a_n &= (q_1^T a_n) q_1 + (q_2^T a_n) q_2 + \cdots + (q_{n-1}^T a_n) q_{n-1} + \|q_n'\| q_n \end{aligned}$$

Bestimmt man also die Matrix $Q$ aus Gram-Schmidt und die Matrix $R$ aus den Gleichungen oben, so ergibt sich für $A \in \mathbb{R}^{m \times n}$ mit $m > n$ die QR-Faktorisierung $A = Q \cdot R$ mit $Q \in \mathbb{R}^{m \times n}$, $R \in \mathbb{R}^{n \times n}$.

**Eigenschaften**

- Die QR-Zerlegung existiert, wenn $m \geq n$ und $A$ vollen Rang $n$ hat.
- Ansonsten benutzt man die Singulärwertzerlegung, siehe Abschnitt 8.4.
- Die **Normalengleichung** wird ja nun zu

  $$A^T A x = A^T b \quad \Leftrightarrow \quad R^T Q^T Q R x = R^T Q^T b \quad \Leftrightarrow \quad R x = Q^T b.$$

  Die Lösung $x$ berechnet sich also durch

  1. Berechne $Q$ und $R$,
  2. berechne $y = Q^T b$,
  3. löse $Rx = y$ durch Rückwärtssubstitutieren.

- Die Kondition der Normalengleichung ist nun nur noch $\kappa(A)$ statt $\kappa(A)^2$, denn (zumindest für die Spektralnorm $\|\cdot\|_2$) gilt:

  $$\kappa(R) = \kappa(QR) = \kappa(A).$$

- Für die Pseudoinverse $A^+ = (A^T A)^{-1} A^T b$ gilt: $A^+ = R^{-1} Q^T$.
- Der Rechenaufwand ist etwa $2mn^2$ FLOP (Gram-Schmidt) bzw $2mn^2 - \frac{2}{3}n^2$ FLOP (mit Householdertransformationen, siehe nächster Abschnitt).
- Also kann man die QR-Zerlegung auch nutzen zum Lösen von $m \times m$ Gleichungssystemen (Aufwand doppelt so groß wie LR-Zerlegung).

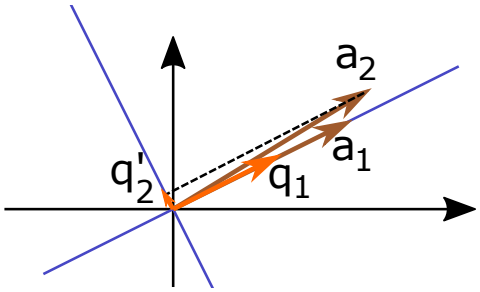



Figure 9: Beim Anwenden von Gram-Schmidt entsteht der Vektor $q_2'$ durch Projizieren von $a_2$ auf das orthogonale Komplement von $q_1$. Das Ergebnis ist ein sehr kurzer Vektor $q_2'$.

## 8.3 QR mit Householdermatrizen

Aus Gründen numerischer Stabilität und Effizienz ist Gram-Schmidt nicht immer gut. Wann ist das nicht gut? Abbildung 9 zeigt eine solche Situation: wenn die Vektoren $a_1$ und $a_2$ einen sehr kleinen Winkel bilden (also die Konditionszahl der Matrix $(a_1|a_2)$ groß ist). Ein kleines Wackeln an $a_2$ bewirkt dann eine (prozentual) große Änderung am projizierten Vektor $q_2'$.

Daher ist die Methode der Householder-Transformationen vorzuziehen, die ist immer stabil. Wir geben hier nur die Idee wieder (siehe auch Abbildung 10).

Statt $a_1$ einfach nur zu $q_1$ zu normieren, "projizieren" wir nun jeweils alle $a_1, a_2, \ldots, a_n$ auf die Koordinatenachsen, also auf die $\langle e_1\rangle, \langle e_2\rangle, \ldots, \langle e_n\rangle$ (wobei $e_1, e_2, \ldots, e_m$ die Einheitsvektoren $e_1 = (1,0,\ldots,0)^T$ usw bezeichnet). Statt $a_i$ orthogonal darauf zu projizieren, spiegeln wir $a_i$ an der Winkelhalbierenden zwischen $a_i$ und $e_i$. Das heißt

- Sei $w_i' = a_i - q_i' = a_i - \|a_i\| e_i$, und $w_i = \frac{1}{\|w_i'\|} w_i'$.
- Die Matrix $H_i := E - 2 w_i w_i^T$ spiegelt $a_i$ auf $\langle e_i\rangle$.

Abbildung 10 zeigt die Wirkung von $H_2$. Es ist ja $E - w_2 w_2^T$ die Projektion auf $w_2^\perp$. Also kann man sich vielleicht vorstellen, das $E - 2 w_2 w_2^T$ den Vektor $a_2$ um die doppelte Strecke bewegt, also eine Spiegelung an $w_2^\perp$ bewirkt.

Nach dem ersten Schritt erhalten wir

$$H_1 \cdot A = \begin{pmatrix} \alpha_1 & \star & \cdots & \star \\ 0 & & & \\ \vdots & & A' & \\ 0 & & & \end{pmatrix}.$$

Wir machen dann analog weiter mit der $(m-1)\times(n-1)$ Matrix $A'$ usw. Am Ende steht rechts ja eine obere Dreiecksmatrix, die ist unser $R$. Das $Q$ erhalten wir analog zur LU-Zerlegung oder QR-Zerlegung als

$$Q = (H_{m-1} \cdots H_2 \cdot H_1)^{-1} = H_1^T \cdot H_2^T \cdots H_{m-1}^T.$$

Das war jetzt nur eine grobe Beschreibung der QR-Zerlegung mit Householdertransformationen, um die Idee deutlich zu machen. Zu den genauen Details siehe z.B. wikipedia.

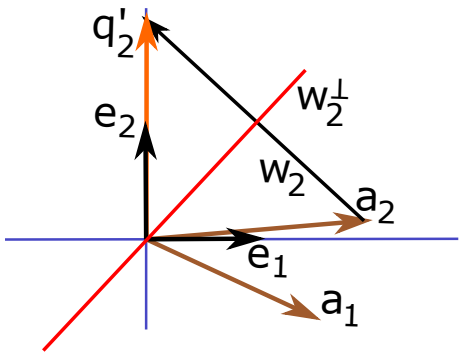



Figure 10: Beim Anwenden von Householdermatrizen entsteht der Vektor $q_2'$ durch Spiegeln von $a_2$ an der Winkelhalbierenden $\langle w_2 \rangle$ von $a_2$ und $e_2$.

## 8.4 Singulärwertzerlegung (SVD)

Der wesentliche Vorteil der QR-Zerlegung ist, dass die Matrix $Q$ aufgrund ihrer orthonormalen Spalten eine einfache Projektion auf das Bild von $A$ ermöglicht. Die Dreiecksmatrix $R$ erlaubt dann ein effizientes Berechnen der Lösung von $Ax = b$ (bzw der Least-Squares-Lösung) durch Rückwärtssubstitution. Die **Singulärwertzerlegung** (singular value decomposition, SVD) geht noch einen Schritt weiter und zerlegt $A$ in noch einfachere Faktoren.

Die Idee ist eine Verallgemeinerung des Konzepts der Eigenwerte bzw der Diagonalisierung von Matrizen. Letzteres klappt selten (z.B. nicht, wenn $A$ nicht quadratisch, oder es gibt komplexe Eigenwerte, oder zu wenige Eigenvektoren,...), die SVD klappt immer. Grundlage ist folgendes Resultat.

**Satz 8.1.** *Für eine Matrix $A \in \mathbb{R}^{m\times n}$ mit $m > n$ existiert die Singulärwertzerlegung (SVD)*

$$A = U\Sigma V^T, \quad \text{wobei } U \in \mathbb{R}^{m\times m} \text{ und } V \in \mathbb{R}^{n\times n} \text{ orthogonal,}$$

$$\text{und } \Sigma \in \mathbb{R}^{m\times n}, \quad \Sigma = \left(\begin{array}{ccc|c} \sigma_1 & & & \\ & \ddots & & 0 \\ & & \sigma_r & \\ \hline & 0 & & 0 \end{array}\right), \quad \text{mit } \sigma_1 \geq \cdots \geq \sigma_r > 0.$$

Falls $A \in \mathbb{R}^{m\times m}$ diagonalisierbar ist, dann ist $\Sigma$ ist die Diagonalmatrix mit den Beträgen $|\lambda_1|, \ldots, |\lambda_m|$ der Eigenwerte $\lambda_1, \ldots, \lambda_m$ von $A$ auf der Hauptdiagonale.

Im Allgemeinen sind die $\sigma_i$ die Quadratwurzeln der Eigenwerte von $A^TA$. Letztere sind ja immer nicht-negativ (warum nochmal?)

Die SVD ist zwar am aufwändigsten zu implementieren, und nicht gerade schnell: sie braucht etwa $2mn^2 + 11n^3$ FLOP. Aber sie ist die "eierlegende Wollmilchsau" unter den LGS-Lösern.

- Lösung der Normalengleichung $A^TAx = A^Tb$ (überbestimmtes LGS):
  1. Berechne die SVD $A = U\Sigma V^T$,
  2. berechne $b' = U^Tb$,

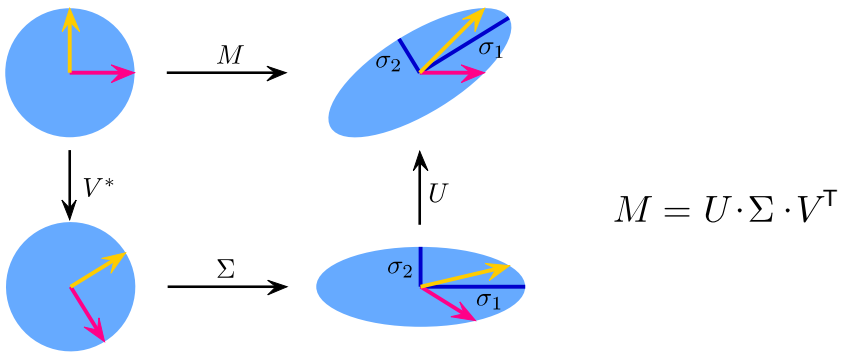



Figure 11: Darstellung einer Scherung als Kombination von Drehen, Skalieren, Drehen.

3. löse $\Sigma y = b'$ (trivial, da Diagonalgestalt),
4. berechne $x = Vy$.

Wie bei der QR-Zerlegung bleibt die Konditionszahl des Problems bei $\kappa(A)$ (statt $\kappa(A)^2$).

- Die Pseudoinverse $A^+$ von $A$ ist $V\Sigma^{-1}U^T$. Dabei ist erstens $\Sigma^{-1}$ trivial wegen Diagonalgestalt. Zweitens ist $\Sigma$ ja nicht unbedingt invertierbar, also ist $\Sigma^{-1}$ so zu verstehen: $\sigma_i > 0$ wird zu $1/\sigma_i$, und $\sigma_i = 0$ bleibt 0.
- Damit kann mittels der SVD auch für $A \in \mathbb{R}^{m\times n}$ mit $\text{Rang}(A) < n$ die Pseudoinverse definiert werden.

Zusätzlich liefert die SVD noch eine interessante geometrische Interpretation einer linearen Abbildung $f : \mathbb{R}^m \to \mathbb{R}^n$, $f(x) = Ax$: Die Matrizen $U$ und $V$ sind orthogonal, beschreiben also Drehungen oder Spiegelungen. Die Diagonalmatrix $\Sigma$ entspricht einer Skalierung des $\mathbb{R}^n$, welche jede Achse $e_i$ um den Faktor $\sigma_i$ streckt/staucht.

Satz 8.1 sagt daher aus: jede linare Abbildung von $\mathbb{R}^m$ nach $\mathbb{R}^n$ ist eine Kombination von Spiegeln oder Drehen, Skalieren entlang der Achsen $e_i$ und nochmal Spiegeln oder Drehen. Das ist in Abbildung 11 illustriert für die Scherung $M = \left(\begin{smallmatrix} 1 & 1 \\ 0 & 1 \end{smallmatrix}\right)$.

# 9 Überblick: Löser für LGS

Wir sahen nun schon einige Löser (d.h., Lösungsmethoden) für LGS, wie sie in der Praxis eingesetzt werden: LU-Zerlegung, Cholesky, QR-Zerlegung und Singulärwertzerlegung. Dies sind gebräuchliche Verfahren für allgemeine LGS. In der Praxis tauchen allerdings auch häufig sehr dünn besetzte LGS auf, d.h. solche, wo fast überall Nullen stehen. Für diese gibt es andere Verfahren, die das ausnutzen und für diese LGS effizienter sind, und die wir hier nicht behandelt haben.

Alle oben genannten Verfahren sind exakte Verfahren, d.h. sie liefern die Lösung im Prinzip beliebig genau (aber...). Daneben gibt es noch approximative Verfahren. Diese liefern nur eine Näherungslösung, sind dafür aber noch schneller, und daher in der Lage, noch größere LGS zu behandeln. Zwei Beispiele sind das/die CG-Verfahren (*conjugate gradients*, konjugierte Gradienten, 1952) oder das/die GMRES-Verfahren (*generalized minimal residual method*, 1986).

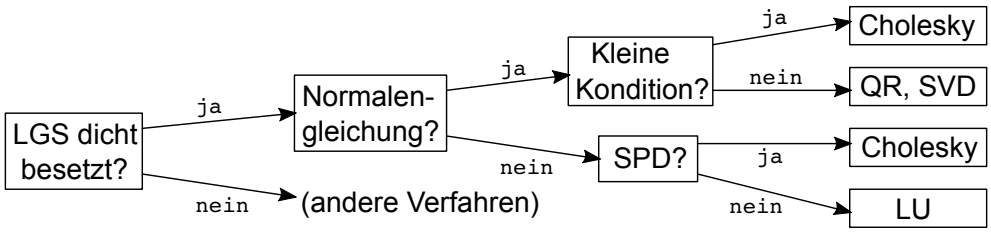



Figure 12: Übersicht

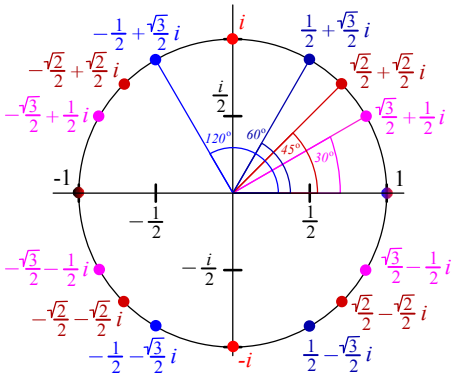



Figure 13: Die komplexen $n$-ten Einheitswurzeln, für $n = 3$ (hellblau), $n = 4$ (knallrot), $n = 6$ (hell- und dunkelblau), $n = 8$ (knallrot und dunkelrot), $n = 12$ (violett und hell- und dunkelblau).

# 10 DFT

Erinnerung: die Menge $\mathbb{C}$ der **komplexen Zahlen** ist $\mathbb{C} = \{a + bi \mid a, b \in \mathbb{R}\}$. Dabei ist $i^2 = -1$. Ein paar wichtige Begriffe und Konzepte:

- Das **konjugiert komplexe** $\bar{x}$ von $x = a + bi$ ist $\bar{x} = a - bi$.
- Es gilt die Eulersche Identität: $e^{ix} = \cos(x) + i\sin(x)$.
- Die ($N$-ten) **komplexen Einheitswurzeln** sind die Lösungen von $x^N = 1$. Für $N = 2$ sind das gerade 1 und $-1$ (langweilig!), aber für $N > 2$ wird die Sache interessant, siehe Abbildung 13. Wegen der Eulerschen Identität sind die $N$-ten Einheitswurzeln $e^{2\pi ik/N}$ ($k = 0, 1, \ldots, N-1$).

Angenommen, wir betrachten Funktionen, die als Definitionsbereich nicht $\mathbb{R}$ oder $[-\pi, \pi]$ haben, sondern welche mit einem endlichen Definitionsbereich:

$$f : \{0, 1, \ldots, N-1\} \to \mathbb{C}$$

So eine Funktion kann man sich einfach als $N$-Vektor vorstellen: Die Funktion ist komplett beschrieben durch die Angabe all ihrer Funktionswerte, das sind $N$ Stück. Also schreiben wir auch einfach $f = (f_0, f_1, \ldots, f_{N-1})$, mit $f_n = f(n)$. Dafür können wir **diskrete Fouriertransformation** (DFT) definieren.

**Definition 10.1.** Die DFT von $f = (f_0, f_1, \ldots, f_{N-1})$ ist $d = (d_0, d_1, \ldots, d_{N-1})$ mit

$$d_k = \frac{1}{N} \sum_{j=0}^{N-1} f_j e^{-2\pi i jk/N} \quad (k = 0, 1, \ldots, N-1)$$

Schreibweise: $\widehat{f} = d$, oder $DFT(f) = d$. Je nach Kontext schreiben wir statt $d$ auch $\widehat{f} = (\widehat{f_0}, \widehat{f_1}, \ldots, \widehat{f_{N-1}})$. Guckt man sich die Formel an, so sieht man, dass da ein Matrix-Vektor-Produkt steht (mit $\xi = e^{2\pi i/N}$) :

$$d = \begin{pmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_{N-1} \end{pmatrix} = \frac{1}{N} \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \xi^{-1} & \xi^{-2} & \ldots & \xi^{-(N-1)} \\ 1 & \xi^{-2} & \xi^{-4} & \ldots & \xi^{-2(N-1)} \\ 1 & \xi^{-3} & \xi^{-6} & \ldots & \xi^{-3(N-1)} \\ \vdots & \vdots & & \ddots & \vdots \\ 1 & \xi^{-(N-1)} & \xi^{-2(N-1)} & \ldots & \xi^{-(N-1)^2} \end{pmatrix} \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{N-1} \end{pmatrix} = \frac{1}{\sqrt{N}} \overline{V} f,$$

wobei $V = \frac{1}{\sqrt{N}} \left(\xi^{jk}\right)_{0 \le j,k \le N-1}$. Den Vorfaktor $\frac{1}{N}$ haben wir in zweimal $\frac{1}{\sqrt{N}}$ aufgespalten, damit gilt:

**Satz 10.2.** *Die Matrix $V$ ist unitär, d.h. $V\overline{V}^T = E$.*

Dabei bezeichnet $E$ die $N \times N$-Einheitsmatrix.

Erinnerung: Eine Matrix $M \in \mathbb{R}^{N \times N}$ heißt *orthogonal*, falls $MM^T = E$. In $\mathbb{R}^2$ oder $\mathbb{R}^3$ sind das gerade die Matrizen, die eine Drehung oder eine Spiegelung bewirken. Orthogonale Matrizen bewirken abstands- und winkeltreue Abbildungen. Für das Standardskalarprodukt $v^T \cdot v$ in $\mathbb{R}^d$ drückt sich das so aus: $V$ ist orthogonal, genau dann wenn

$$\|Vx\|^2 = (Vx)^T \cdot Vx = x^T \cdot x = \|x\|^2.$$

Also: "Die Länge von $Vx$ ist gleich der Länge von $x$".

Eine Matrix $M \in \mathbb{C}^{N \times N}$ heißt *unitär*, falls $M\overline{M}^T = E$. Für relle Matrizen heißt das dasselbe wie orthogonal. Für das Standardskalarprodukt im $\mathbb{C}^d$ drückt sich das so aus: $V$ unitär, genau dann wenn $(Vx)^T \cdot \overline{Vx} = x^T \cdot \bar{x}$.

*Proof.* Wir zeigen das zeilenweise. Es muss gelten: $k$-te Zeile mal $k$-te Zeile = 1, und $k$-te Zeile mal $m$-te Zeile = 0 für $m \neq k$. Bezeichnen wir mit $V_k$ die $k$-te Zeile von $V$. Dann gilt:

$$V_k(\overline{V_k})^T = \frac{1}{N} \sum_{\ell=0}^{N-1} \xi^{\ell k} \overline{\xi^{\ell k}} = \frac{1}{N} \sum_{\ell=0}^{N-1} \xi^{\ell k} \xi^{-\ell k} = \frac{1}{N} \sum_{\ell=0}^{N-1} \xi^{(\ell-\ell)k} = \frac{1}{N} \sum_{\ell=0}^{N-1} 1 = 1,$$

und für $k \neq m$:

$$V_k(\overline{V_m})^T = \frac{1}{N} \sum_{\ell=0}^{N-1} \xi^{\ell k} \xi^{-\ell m} = \frac{1}{N} \sum_{\ell=0}^{N-1} (\xi^{k-m})^{\ell} = \frac{1}{N} \frac{\xi^{(k-m)N} - 1}{\xi^{k-m} - 1}.$$

Der Zähler im letzten Term ist $1 - 1 = 0$, da $\xi$ eine komplexe $N$-te Einheitswurzel ist [WIK]. Der Nenner ist ungleich null, da $-N < k - m < N$ und $k - m \neq 0$; aber $\xi^n$ wird nur 1, falls $n$ Vielfaches von $N$ ist. Also ist für $m \neq k$

$$V_k(\overline{V_m})^T = 0$$

und die Behauptung ist bewiesen. □

**Bemerkung 10.3.** Wegen dieses Satzes ist $V^{-1} = \overline{V}^T = \overline{V}$. Die Inverse von $V$ ist also effizient berechenbar.

Die Umkehrung der DFT, die *inverse DFT*, ist definiert durch

$$IDFT(d) = \check{d} = \sqrt{N} V d$$

Damit bekommt man (natürlich!) aus jedem $d = \widehat{f}$ das $f$ zurück. Hier sind's ja nur Matrix-Vektor-Multiplikationen, um Existenz von Integralen braucht man sich keine Sorgen zu machen.

**Satz 10.4.** *Für $V$ wie oben gilt: $V^4 = E$, wobei $E$ die Einheitsmatrix bezeichnet.*

Daraus folgt direkt: $\boxed{\widehat{\widehat{\widehat{f}}} = \frac{1}{N^2} f}$

Also liefert (wie auch später im kontinuierlichen Fall) die viermalige Anwendung der Fouriertransformation das $f$ zurück, bis auf einen Vorfaktor (später $2\pi$, hier eben $\frac{1}{N^2}$).

Weiterhin gelten etliche weitere Beziehungen (später sehen wir deren Entsprechungen auch im kontinuierlichen Fall). Zum Beispiel gelten für $f = (f_0, f_1, \ldots, f_{N-1}) \in \mathbb{C}^N$, $g = (g_0, g_1, \ldots, g_{N-1}) \in \mathbb{C}^N$ und deren DFTs $\widehat{f} = (\widehat{f_0}, \widehat{f_1}, \ldots, \widehat{f_{N-1}})$, $\widehat{g} = (\widehat{g_0}, \widehat{g_1}, \ldots, \widehat{g_{N-1}})$ folgende Aussagen:

1. **Faltungssatz:** $\widehat{f * g} = N \widehat{f} \cdot \widehat{g}$. Dabei ist

   $$f * g(n) = \frac{1}{N} \sum_{k=0}^{N-1} f_k g_{n-k} \qquad (0 \le n \le N-1)$$

   Der Index $n-k$ von $g_{n-k}$ kann negativ werden, also legen wir hier für $1 \le k \le N-1$ fest: $g_{-k} = g_{N-k}$.

2. **Satz von Parseval:**

   $$\sum_{n=0}^{N-1} |d_n|^2 = \frac{1}{N} \sum_{n=0}^{N-1} |f_n|^2$$

3. **Satz von Plancherel:**

   $$\sum_{n=0}^{N-1} \widehat{f_n} \overline{\widehat{g}_n} = \frac{1}{N} \sum_{n=0}^{N-1} f_n \overline{g}_n$$

4. **Darstellung als FR:** Die FR von $f$ beschreibt $f$:

   $$f_n = \sum_{k=0}^{N-1} \widehat{f_k} e^{2\pi i k n/N} = \sum_{k=0}^{N-1} \left( \widehat{f_k} \cos(2\pi i k n/N) + \widehat{f_k} \sin(2\pi i k n/N) \right)$$

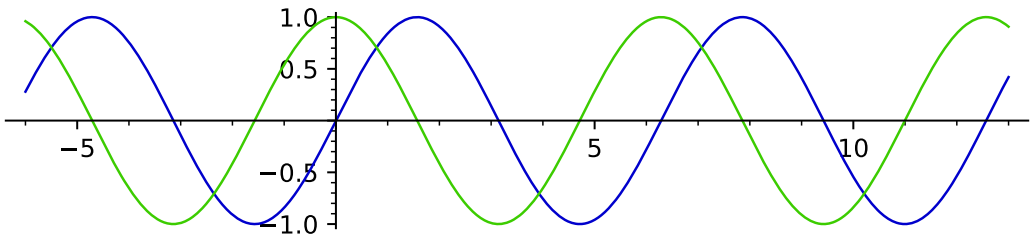
Zu den Beweisen der ersten beiden Aussagen: Übung. Bezüglich der letzten stellen wir fest: Es gilt $\widehat{f} = \frac{1}{\sqrt{N}} \overline{V} f$, also

$$f = \sqrt{N} \overline{V}^{-1} \widehat{f} = \sqrt{N} V \widehat{f},$$

das heißt für den $n$-ten Eintrag $f_n = \sum_{k=0}^{N-1} \widehat{f_k} e^{2\pi i k n/N}$.

**Definition 10.5.** Gegeben eine Funktion $f : \mathbb{R} \to \mathbb{R}$. $f$ heißt *gerade* (bzw *ungerade*), falls $f(-x) = f(x)$ (bzw. falls $-f(-x) = f(x)$).

**Beispiel 10.6.** Die Sinusfunktion ist $2\pi$-periodisch und ungerade (blauer Graph):




Die Cosinusfunktion ist $2\pi$-periodisch und gerade (grüner Graph).

**Satz 10.7.** *Jede Funktion $f : \mathbb{R} \to \mathbb{R}$ (oder $f : [-a,a] \to \mathbb{R}$) lässt sich als Summe $g+h$ einer geraden Funktion $g$ und einer ungeraden Funktion $h$ darstellen.*

*Proof.* Sei $g(x) = \frac{1}{2}(f(x)+f(-x))$ und sei $h(x) = \frac{1}{2}(f(x)-f(-x))$.

[ Nachrechnen: $g$ ist gerade, $h$ ist ungerade, $f = g+h$ ]

□

Für die DFT ist das wichtig, denn: falls wir statt $f : \{0,1,\ldots,N-1\} \to \mathbb{C}$ den Definitionsbereich symmetrisch machen, z.B. so:

$$f : \{-k,\ldots,-1,0,1,2,\ldots,k\} \to \mathbb{C}, \quad f(x) = \ldots$$

klappt alles ganz analog (mit $N = 2k+1$). Aber, wegen Punkt 4 oben ("Darstellung als FR") fallen die Sinusterme $\cdots i\sin(\cdot)$ weg. (Das ist an dieser Stelle nicht offensichtlich, nur plausibel. Dazu muss man sich eigentlich mehr überlegen, aber es ist wahr.) Wenn außerdem $f$ reell ist, werden die Koeffizienten $\widehat{f}_k$ auch reell sein. (Dito... plausibel, aber ...) Daher gilt:

**Satz 10.8.** *Die DFT einer geraden Funktion $f : \mathbb{R} \to \mathbb{R}$ enthält nur noch Cosinusterme mit reellen Koeffizienten.*

## 10.1 Diskrete Cosinus-Transformation (DCT)

Bisher war unser $f$ auf $\{0,1,2,\ldots,N-1\}$ definiert. Genausogut können wir das so skalieren, dass wir immer im Intervall $[0,1]$ liegen:

$$f : \{0, \frac{1}{N}, \frac{2}{N}, \ldots, \frac{N-1}{N}\} \to \mathbb{C}.$$

Daher können wir $f$ zu einer geraden Funktion fortsetzen, indem wir $f$ auf $\{\frac{-N+1}{N}, \ldots, \frac{N-1}{N}\}$ definieren durch $f(x) = f(-x)$. Dann werden in obigem Ansatz alle Sinuskoeffizienten zu Null, nur die Cosinusterme überleben, und wir erhalten die (oder besser eine) **Diskrete Cosinus-Transformation (DCT)**. Wir haben aber noch weitere Variationsmöglichkeiten: Eben haben wir die $f_0, f_1, f_2, \ldots f_{N-1}$ als Funktionswerte an den Stellen $0, \frac{1}{N}, \frac{2}{N}, \ldots, \frac{N-1}{N}$ interpretiert. Das ist etwas unsymmetrisch: der linke Rand ist drin, der rechte nicht. Also können wir es auch symmetrischer machen, indem wir die Werte um $\frac{1}{2N}$ verschieben: $f_0, f_1, f_2, \ldots, f_{N-1}$ sind dann die Funktionswerte an den Stellen $\frac{1}{2N}, \frac{1}{N} +$

$\frac{1}{2N}, \frac{2}{N} + \frac{1}{2N}, \ldots, \frac{N-1}{N} + \frac{1}{2N}$. Das liefert zwei Wahlmöglichkeiten. (Bzw vier, da wir die Option "Rand drin" für den rechten und den linken Rand unabhängig treffen können. Aber meist macht man es rechts und links gleich.)

Wir können auch noch festlegen, ob die Funktion nach rechts gerade oder ungerade fortgesetzt wird, siehe Bild 14. Das liefert also vier Ansätze, die DCT zu berechnen.

$$\text{DCT I:} \qquad a_k = \frac{1}{2}(f_0 + (-1)^k f_{N-1}) + \sum_{n=1}^{N-2} f_n \cos\left(\frac{\pi}{N-1} nk\right) \qquad k = 0, \ldots, N-1. \tag{10.1}$$

$$\text{DCT II:} \qquad a_k = \sum_{n=0}^{N-1} f_n \cos\left(\frac{\pi}{N}\left(n+\frac{1}{2}\right)k\right) \qquad k = 0, \ldots, N-1. \tag{10.2}$$

$$\text{DCT III:} \qquad a_k = \frac{1}{2} f_0 + \sum_{n=1}^{N-1} f_n \cos\left(\frac{\pi}{N} n\left(k+\frac{1}{2}\right)\right) \qquad k = 0, \ldots, N-1. \tag{10.3}$$

$$\text{DCT IV:} \qquad a_k = \sum_{n=0}^{N-1} f_n \cos\left(\frac{\pi}{N}\left(n+\frac{1}{2}\right)\left(k+\frac{1}{2}\right)\right) \qquad k = 0, \ldots, N-1. \tag{10.4}$$

Dabei ist häufig DCT II gemeint, wenn von "der DCT" die Rede ist. Die inversen DCTs sehen wie folgt aus:

- Die inverse DCT I ist DCT I mal $\frac{2}{N-1}$.
- Die inverse DCT II ist DCT III mal $\frac{2}{N}$.
- Die inverse DCT III ist DCT II mal $\frac{2}{N}$.
- Die inverse DCT IV ist DCT IV mal $\frac{2}{N}$.

# 11 Bildkompression

Die Idee zur Bildkompression ist diese: Wegen des Satzes von Riemann-Lebesgue (siehe wikipedia) erwarten wir, dass $a_k \to 0$ für $k \to \infty$. Die Werte von $a_k$ werden also klein, wenn $k$ groß ist. Nehmen

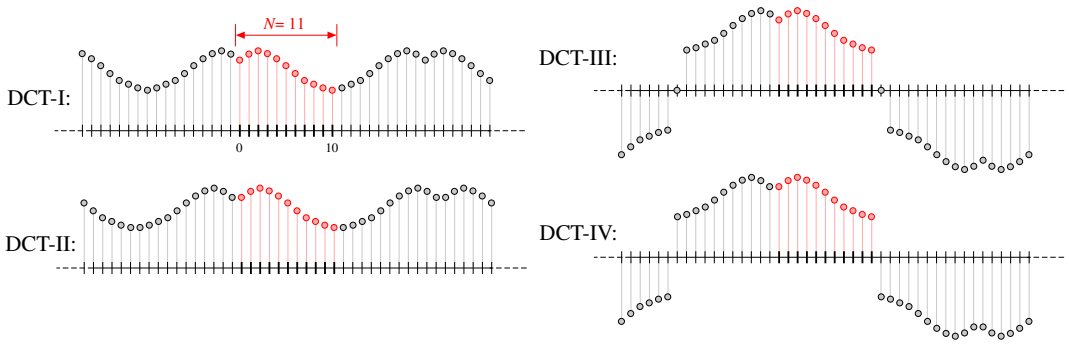



Figure 14: Vier Möglichkeiten zur Definition der DCT. (Quelle: [WIK])

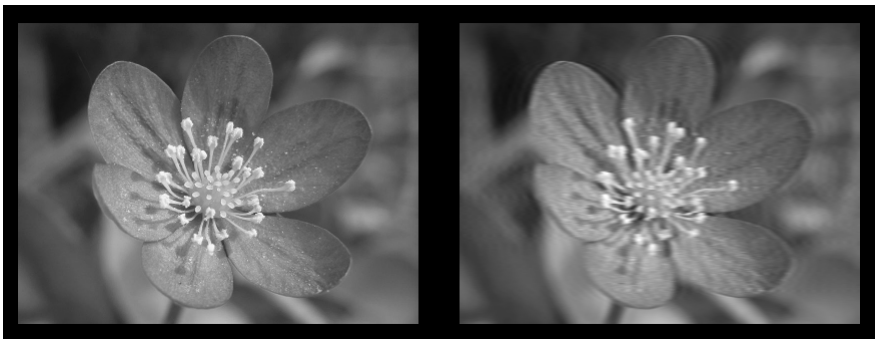


Figure 15: Ein Bild mit 300 mal 400 Pixeln, links das Original, rechts das nach unserer Methode komprimierte. Das Bild rechts ist klar unschärfer, aber das Motiv mit allen Details ist zu erkennen.

wir an, wir haben ein Schwarzweißbild (also eigentlich ein Grauwertebild), dann nehmen wir eine einzelne Bildzeile und interpretieren die Grauwerte des $k$-ten Pixels als Funktionswert $f_k$. (Oft gibt es 256 mögliche Grauwerte). Jede Zeile hat $N$ Pixel, also bekommen wir einen Vektor $f \in \mathbb{R}^N$.

Auf das so erhaltene $f$ wenden wir DCT an. Von $DFT(f)$ speichern wir nur die ersten 20 % (oder so) der Werte. Zum Angucken berechnen wir die inverse DCT, und hoffen, dass (da die letzten 80% nahe an Null sind, also fast Null, also nix ausmachen sollten) das so erhaltene Bild sehr nah am ursprünglichen Bild ist. Das klappt. Ein Spielzeugbeispiel: Unsere Bildzeile sehe so aus:

 $(1,51,101,151,201,151,101,51)$

Rechts steht der Vektor der Grauwerte (0=schwarz, 255=weiß). Darauf wenden wir die DCT II an und erhalten (auf ganze Zahlen gerundet) $(808,-128,-315,45,0,-30,-22,25)$. Setzen wir die letzten vier Werte auf 0 und berechnen die inverse DCT davon, erhalten wir (gerundet)

$$(6,42,102,161,186,160,100,50).$$

Stellen wir das nun wieder als Graubild dar, so sehen wir praktisch keinen Unterschied.

Wieso ist das eine Bild*kompression*? Das, was wir speichern, ist das Ergebnis der DCT II, ohne die Nullen. (Also nur 20% des Speicherbedarfs.) Wenn wir das Bild in einem Bildbetrachter öffnen, dann weiß dieser, dass jeder gespeicherten Zeile 80% Nullen anzuhängen sind. Also kommen wir mit etwa einem Fünftel des eigentlichen Speicherbedarfs aus. Das Ergebnis sehen wir in Abbildungen 15 und 16.

## 11.1 Jpeg

Die Vorgehensweise von `jpeg` ist natürlich etwas komplizierter. Statt pauschal 80% der Werte auf Null zu setzen, wird etwas klüger gerundet. Außerdem gibt es Qualitätsprobleme, falls in einer Zeile "hohe Frequenzen" auftauchen, das heißt hier: häufiger Wechsel von Hell zu Dunkel u.U. in kurzen Abständen (vgl. das Wellenmuster in Abbildung 16 rechts). Man beachte auch, dass die obersten Bidzeilen (einfarbig grau) keine Fehler aufweisen. Auch diese Probleme werden in `jpeg` klug gelöst.

Die Schritte in `jpeg` zum Komprimieren eines Farbbilds:

- Bearbeite jeden der drei Farbwerte einzeln (RGB bzw. YCbCr)

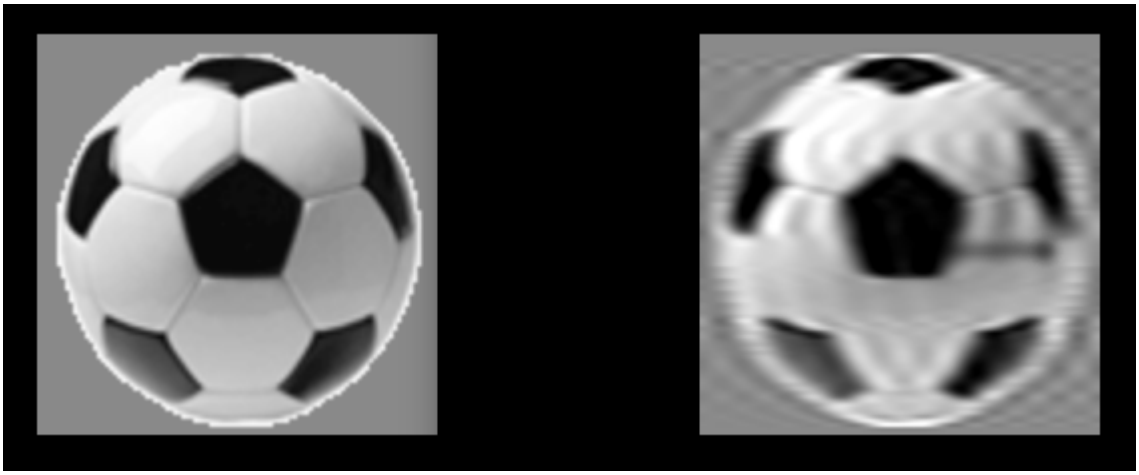


Figure 16: Ein Bild mit nur 100 mal 100 Pixeln, links das Original, rechts das nach unserer Methode komprimierte. Es gibt deutliche Fehler: die schwarzen Flecken links oben auf dem Ball, sowie eine Art Wellenmuster, das das Bild überlagert (außer in den grauen Zeilen ganz oben!).

- Unterteile das Bild in $8 \times 8$ Felder (evtl am Rand mit Nullen füllen)
- DCT für jedes einzelne Feld, erst zeilen- dann spaltenweise
- Quantisieren der DCT (s.u., dies ist die einzige Stelle, wo Information verloren geht)
- Run-length encoding
- Huffman encoding

Ergebnis ist je eine Integer-Sequenz für jedes $8 \times 8$-Feld. Deren Länge liegt deutlich unter 256 Byte. Diese Sequenzen werden als `jpeg`-file gespeichert. Beim Angucken eines jpg-files werden die umgekehrten Schritte ausgeführt: Huffman-encoding rückwärts, run-length-encoding rückwärts (beides verlustfrei), IDCT, zusammensetzen der $8 \times 8$-Felder und der drei Farbwerte zum Gesamtbild.

Zu den einzelnen Schritten: die ersten beiden bedürfen keiner weiteren Erklärung. Die Unterteilung in $8 \times 8$ Felder verhindert eine Fehlerausbreitung wie in dem Wellenmuster in Abbildung 16: eventuelle Welleneffekte sind auf ein $8 \times 8$-Feld begrenzt.

Bei dem dritten Schritt machen wir in der Tat mehrere DCT II auf einer $8 \times 8$ Matrix, und zwar einmal zeilenweise (jede der 8 Zeilen einzeln), und dann für das Ergebnis davon spaltenweise (jede der 8 Spalten einzeln). Wir erhalten so wieder eine $8 \times 8$ Matrix.

**Quantisieren** heißt hier: Die Einträge der Matrix werden auf die nächste ganze Zahl gerundet, aber bezüglich einer Quantisierungsmatrix $Q$: Der Eintrag an der Stelle $k,m$ wird durch $Q_{k,m}$ geteilt und dann gerundet. Unten ein Beispiel: Das erste ist die DCT für ein Feld. Der Eintrag oben links, also -415, wird durch 16 geteilt: -25,9375, und dann gerundet: -26. Die Matrix $Q$ ist so gewählt, dass links oben, wo die Koeffizienten mit niedrigem Index stehen, relativ genau gerundet wird, und unten rechts, wo die Koeffizienten mit hohem Index stehen, recht grob gerundet wird. Hier eine typische $8 \times 8$-Matrix, wie sie nach der DCT entstehen könnte.

$$DCT: \begin{bmatrix} -415 & -30 & -61 & 27 & 56 & -20 & -2 & 0 \\ 4 & -22 & -61 & 10 & 13 & -7 & -9 & 5 \\ -47 & 7 & 77 & -25 & -29 & 10 & 5 & -6 \\ -49 & 12 & 34 & -15 & -10 & 6 & 2 & 2 \\ 12 & -7 & -13 & -4 & -2 & 2 & -3 & 3 \\ -8 & 3 & 2 & -6 & -2 & 1 & 4 & 2 \\ -1 & 0 & 0 & -2 & -1 & -3 & 4 & -1 \\ 0 & 0 & -1 & -4 & -1 & 0 & 1 & 2 \end{bmatrix}$$

Deren Einträge runden wir bzgl der Quantisierungsmatrix $Q$:

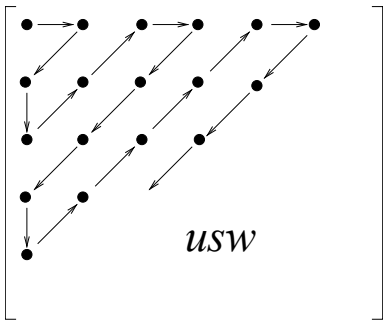
$$Q = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

Das ergibt in diesem Beispiel:

$$\begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -4 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Wenn der Rechner jemanden fragt, wie hoch der Kompressionsgrad sein soll, so wählt man damit eine gröbere (hohe Kompression) bzw feinere Matrix $Q$ aus. Diese Matrizen sind feste Bestandteile von `jpeg`.

**Run-length enconding (RLE).** Nach dem Quantisieren steht da eine Matrix, die tendenziell unten rechts nur Nullen enthält. Auf diese Matrix wenden wir RLE an, indem wir sie diagonal durchlaufen, nach dem folgenden Schema.



RLE ist so ein einfaches Konzept, dass ein Beispiel genügen soll: Wollen wir die Zeichenkette

000000000000000000011111111111111111100000000000011111000000000

naiv speichern, so schreiben wir achtzehn Nullen hintereinander, dann achtzehn Einsen, dann zwölf Nullen, dann fünf Einsen, und zuletzt neun Nullen. Also insgesamt 62 Zeichen bzw Byte.

Speichern wir das aber als 18 0en, 18 1en, 12 0en, 5 1en, 9 0en, also etwa als

$$(18,0,18,1,12,0,5,1,9,0),$$

so sind das im Idealfall nur 10 statt 62 Zahlen bzw Zeichen. Eventuell brauchen wir Trennzeichen (die Kommas) und ein Endezeichen, sowie mehr als ein Zeichen pro Zahl (z.B. 18), aber dann sind es hier immer noch nur 23 statt 62 Zeichen.

Obwohl die Idee so simpel ist, ist RLE wichtiger Bestandteil von `jpeg`. Gerade hier (nach dem Runden) kommt das Ergebnis kommt mit bedeutend weniger Speicherplatz aus als das ursprüngliche Bild.

**Huffman Encoding.** Das Problem bei RLE ist noch, dass wir eine 0 mit einem ganzen Byte speichern (8 bit), und für eine 12 sogar mit zwei Byte (16 bit). Wenn wir wissen, dass unsere Zeichenkette nur wenige Zeichen enthält (die zehn Ziffern und ","), dann geht es viel besser.

Die Idee beim Huffman-Coding ist, häufige Zeichen mit wenigen bit zu speichern. Insbesondere brauchen wir nur wenige bit pro Zeichen, falls insgesamt wenige verschiedene Zeichen vorkommen.

Die genauere Idee ist, alle vorkommenden Zeichen mit ihren Häufigkeiten als die Blätter eines noch zu konstruierenden Binärbaums zu betrachten. Angefangen bei den seltensten Blättern (Zeichen) werden je zwei Blätter mit einem neuen Knoten (als Vorfahr) verbunden. Dieser erhält die Summe der Häufigkeiten der beiden Kinder als Eintrag. Repeat.

Ist der Baum fertig, so wird jedes Blatt als 0-1-Wort kodiert, wobei das 0-1-Wort dem Pfad von der Wuuzel zu ihm entspricht. (0: rechts 1: links). Wieder mal wird das ganze vielleicht an einem Beispiel am klarsten.

**Beispiel 11.1.** : Nachricht: 11112233455555666666.

Naiv brauchen wir 20 Zeichen, also 160 bit.

Etwas cleverer wäre: es gibt hier nur sechs verschiedene Zeichen. Es sind also nur drei Bits pro Zeichen nötig. Also $20 \cdot 3 = 60$ bit Speicherbedarf (plus einen konstanten Anteil für Metadaten oder Syntax).

Huffman-Kodieren ist noch effizienter. Dazu erstellen wir den Baum in Abbildung 17. Damit wird 11112233455555666666 so gespeichert:

●○|●○|●○|●○|●○○|●○○|●○○○|●○○○|○○○○|●●|●●|●●|●●|●●|○●|○●|○●|○●|○●|○●|

Hier brauchen wir — trotz der verschiedenen Längen — keine Trennzeichen! (Die | müssen also nicht gespeichert werden.) Wir lesen immer, bis wir an einem Blatt ankommen. Danach beginnt ein neues Zeichen.

Der Speicherbedarf ist

$$4 \cdot 2 + 2 \cdot 3 + 2 \cdot 4 + 1 \cdot 4 + 5 \cdot 2 + 6 \cdot 2 = 48$$

Also 48 bit Speicher, statt 60 bit. Das sind im Schnitt 2,4 Bit pro Zeichen.

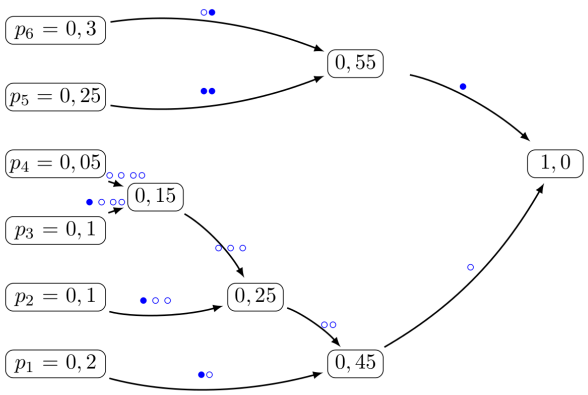



Figure 17: Der Baum zur Huffman-Kodierung. Wir starten mit den Blättern (links). Die seltensten Zeichen bzw Knoten liefern einen Vorfahr. Der erhält die Summe der Häufigkeiten der beiden Kinder. Hier beginnt es so: als erstes liefern 3 und 4 einen neuen Elternknoten (mit Eintrag 0,15). Im nächsten Schritt wird dieser mit 2 zu einem weiteren neuen Knoten (mit Eintrag 0,25), usw. *(Sorry, im Bild sind die Zuweisungen 1-oben und 0-unten noch durcheinander.)*

# 12 FFT

Offenbar ist es wünschenswert, die DFT oder DCT effizient zu berechnen. Das geht so.

**Satz 12.1.** *Aus den DFTs von* $(f_0, f_1, \ldots, f_{N-1})$ *und* $(f_N, f_{N+1}, \ldots, f_{2N-1})$ *erhalten wir die DFT von* $DFT(f_0, f_N, f_1, f_{N+1}, f_2, f_{N+2}, \ldots, f_{N-1}, f_{2N-1})$ *(Länge 2N) so: Mit*

$$d_k = \frac{1}{2}\big(DFT(f_0, f_1, \ldots, f_{N-1}) + e^{-\pi ik/N} DFT(f_N, f_{N+1}, \ldots, f_{2N-1})\big)_k \quad (0 \le k \le N-1)$$

$$d_{N+k} = \frac{1}{2}\big(DFT(f_0, f_1, \ldots, f_{N-1}) - e^{-\pi ik/N} DFT(f_N, f_{N+1}, \ldots, f_{2N-1})\big)_k \quad (0 \le k \le N-1)$$

*ist dann*

$$DFT(f_0, f_N, f_1, f_{N+1}, f_2, f_{N+2}, \ldots, f_{N-1}, f_{2N-1}) = (d_0, d_1, \ldots, d_{2N-1})$$

*Proof.* Für den $k$-ten Eintrag $(0 \le k \le N-1)$ gilt:

$$\begin{aligned} d_k &= \frac{1}{2N}\Big(\sum_{n=0}^{N-1} f_n e^{-2\pi i(2n)k/2N} + \sum_{n=0}^{N-1} f_{N+n} e^{-2\pi i(2n+1)k/2N}\Big) \\ &= \frac{1}{2N}\Big(\sum_{n=0}^{N-1} f_n e^{-2\pi ink/N} + e^{-\pi ik/N} \sum_{n=0}^{N-1} f_{N+n} e^{-2\pi ink/N}\Big) \\ &= \frac{1}{2}\big(DFT(f_0, f_1, \ldots, f_{N-1}) + e^{-\pi ik/N} DFT(f_N, f_{N+1}, \ldots, f_{2N-1})\big)_k \end{aligned}$$

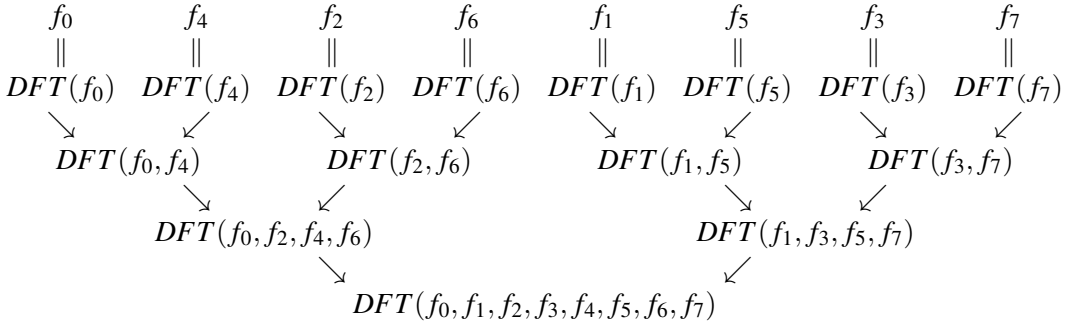
(1. Gleichung: Def DFT, und Aufteilen in zwei Summen, in einer die geraden $(2n)$, in einer die

ungeraden $(2n+1)$.) Für den $k+N$-ten Eintrag ($0 \le k \le N-1$) erhalten wir ganz analog

$$\begin{aligned}
d_{k+N} &= \frac{1}{2N}\Big(\sum_{n=0}^{N-1} f_n e^{-2\pi i(2n)(k+N)/2N} + \sum_{n=0}^{N-1} f_{N+n} e^{-2\pi i(2n+1)(k+N)/2N}\Big) \\
&= \frac{1}{2N}\Big(\sum_{n=0}^{N-1} f_n e^{-2\pi i n(k+N)/N} + e^{-\pi i(k+N)/N} \sum_{n=0}^{N-1} f_{N+n} e^{-2\pi i(k+N)n/N}\Big) \\
&= \frac{1}{2N}\Big(\sum_{n=0}^{N-1} f_n e^{-2\pi i nk/N} e^{-2\pi i n} + e^{-\pi i} e^{-i\pi k/N} \sum_{n=0}^{N-1} f_{N+n} e^{-2\pi i kn/N} e^{-2\pi i n}\Big) \\
&= \frac{1}{2N}\Big(\sum_{n=0}^{N-1} f_n e^{-2\pi i nk/N} - e^{-i\pi k/N} \sum_{n=0}^{N-1} f_{N+n} e^{-2\pi i kn/N}\Big) \\
&= \frac{1}{2}\big(DFT(f_0, f_1, \ldots, f_{N-1}) - e^{-\pi i k/N} DFT(f_N, f_{N+1}, \ldots, f_{2N-1})\big)_k
\end{aligned}$$

□

Dieses Ergebnis liefert einen divide-and-conquer-Algorithums zum Berechnen der DFT in $O(n \log n)$ Schritten. (Hier nur für $N = 2^q$.) Dazu muss das Problem aufgeteilt werden in das Berechnen zweier DFTs, die dann wieder aufgeteilt werden in 2 mal 2 usw.; bis jeweils $N$ Stück DFTs der Länge 1 berechnet werden müssen. Die müssen dann in der richtigen Reihenfolge kombiniert werden. Hier das Schema (für $N = 8$):




Das einzige Problem ist nun, wie bringen wir die $f_n$ in die richtige Anfangsreihenfolge? Die richtige Reihenfolge erhalten wir einfach durch Bitumkehr:

$$000 \to 000,\ 001 \to 100,\ 010 \to 010,\ 011 \to 110,\ 100 \to 001,\ usw$$

Also (im Falle $N = 8$) muss an der 0-ten Stelle $f_0$ stehen, an der ersten Stelle $f_4$, an der zweiten $f_2$ usw. Natürlich hängt die Bitumkehr von $N = 2^q$ ab. Für $N = 16$ ergibt sich etwa

$$0000 \to 0000,\ 0001 \to 1000,\ 0010 \to 0100,\ 0011 \to 1100,\ 0100 \to 0010,\ usw$$

**Algorithmus (FFT):** Sei $N = 2^q$. Sei $g = (g_0, g_1, \ldots, g_{N-1})$ der Vektor, den man durch Umnummerierung mittels Bitumkehr aus $f = (f_0, f_1, \ldots, f_{N-1})$ erhält.

Starte mit den Vektoren der Länge 1: $d^{[0,j]} = (g_j)$ $(j = 0, \ldots, N-1)$. Berechne in Schritt $r$ $(r \ge 1)$ die $2^{q-r}$ Vektoren der Länge $2^r$

$$d^{[r,0]}, d^{[r,1]}, \ldots, d^{[r,2^{q-r}-1]}.$$

aus den Vektoren im $r-1$-ten Schritt gemäß

$$d_k^{[r,j]} = \frac{1}{2}\left(d_k^{[r-1,2j]} + e^{-\pi ik/2^{r-1}} d_k^{[r-1,2j+1]}\right) \tag{12.1}$$

$$d_{2^{r-1}+k}^{[r,j]} = \frac{1}{2}\left(d_k^{[r-1,2j]} - e^{-\pi ik/2^{r-1}} d_k^{[r-1,2j+1]}\right) \tag{12.2}$$

Dabei läuft (innerste Schleife) $k = 0,1,\ldots,2^{r-1}-1$, und (zweitinnerste Schleife) $j = 0,1,\ldots,2^{q-r}-1$, sowie $r = 1,2,\ldots,q$.

**Bemerkung 12.2.** Der obige Algorithmus erlaubt mit wenigen Änderungen auch die Berechnung der IDFT: Der Faktor $\frac{1}{2}$ in (6.5) und (6.6) fällt weg, und aus $e^{-\pi ik/2^{r-1}}$ wird in beiden Gleichungen $e^{\pi ik/2^{r-1}}$.

**Aufwandsbetrachtungen:** Die Matrixmethode braucht $O(N^2)$ Rechenoperationen: Mindestens schon $\frac{1}{2}(N-1)^2$ Multiplikationen, und $N^2$ Additionen. Andererseits gilt: Der FFT-Algorithmus (für $N = 2^q$) braucht $q = \log N$ Schritte (große Schritte, $r$ läuft), und in jedem Schritt $N$ Additionen und $N/2$ Multiplikationen. Also gilt:

| Der FFT- Algorithmus berechnet die DFT von $f$ in $O(N \log N)$ Schritten. |
|---|

(Das Umordnen durch Bitumkehr dürfen wir vernachlässigen, das ist für große $N$ billig: $O(N)$ oder weniger.) Die folgende Tabelle gibt einen Überblick über das Verhältnis des Aufwands beider Algorithmen (aus [AHKKLS]):

| | Aufwand | | |
|---|---|---|---|
| N | normale DFT | FFT | prozentual |
| 4 | 28 | 12 | 42,857% |
| 16 | 496 | 96 | 19,355 % |
| 64 | 8128 | 576 | 7,087% |
| 256 | 130816 | 3072 | 2,348% |
| 1024 | 2096128 | 15360 | 0,733% |

Auch falls $N$ keine Zweierpotenz ist, lassen sich gute Verfahren angeben. Im Wesentlichen geht es nur darum, $N = n\tilde{N}$ in $n$ kleinere Datensätze der Länge $\tilde{N}$ aufzuteilen. Immer wenn $N$ viele kleine Primfaktoren besitzt, geht das gut. Falls $N$ große Primfaktoren besitzt, muss man auf andere Algorithmen zurückgreifen.

# 13 Schnelle Multiplikation

Eine Anwendung der FFT ist schnelle Multiplikation von großen Zahlen. Wir schildern hier ein Spielzeugbeispiel, dass das Prinzip in weiten Teilen verdeutlicht. Der wirkliche Algorithmus ist der *Algorithmus von Schönhage-Strassen*. Der arbeitet in endlichen Zahlringen: $Z_N = \{0,1,2,\ldots,N-1\}$ mit Addition und Multiplikation mod $N$. Auch dort gibt es $n$-te Einheitswurzeln (Lösungen von $x^n - 1 = 0$). Das werden wir hier nicht vertiefen. Die Grundidee wird durch die hier vorgestellte Methode illustriert. Zunächste betrachten wir aber zwei andere Methoden zur Multiplikation: die normale aus der Schule, und den Karatsuba-Algorithmus.

## 13.1 Schulmethode

Wenn wir zwei $n$-stellige Zahlen multiplizieren, müssen wir jede Ziffer der ersten mit jeder Ziffer der zweiten multiplizieren:

| 1 | 2 | 3 | · | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| | 4 | 5 | 6 | | | |
| | | 9 | 1 | 2 | | |
| | | 1 | 3 | 7 | 8 | |
| | 5 | 6 | 0 | 8 | 8 | |

Und dann noch ein paar Additionen durchführen. Bisher hatten wir Additionen und Multiplikationen als gleich teuer aufgefasst. Für Ziffern stimmt das ja auch (binär: XOR ist im Wesentlichen plus, AND ist im Wesentlichen mal, Überträge müssen aber noch berücksichtigt werden).

Schon an diesem Beispiel wird aber doch klar, das Multiplikationen langer Zahlen $O(n^2)$ Elementaroperationen (hier: Multiplikationen von Ziffern) bedeutet, während die Addition nur $O(n)$ Elementaroperationen (hier: Additionen von Ziffern) braucht.

## 13.2 Karatsuba-Algorithmus

Daher ist die Idee beim Karatsuba-Algorithmus diese: Wir führen die Multiplikation zweier Zahlen $x$ und $y$ der Länge $2m$ auf nicht 4 Multiplikationen von Zahlen der Länge $m$ zurück, sondern auf nur 3 Multiplikationen. Wir handeln uns dafür mehr Additionen von $m$-stelligen Zahlen ein, aber das macht uns nichts aus, wegen des oben Gesagten. Das liefert dann einen divide-and-conquer Algorithmus.

Sei $B$ also die Basis unseres Zahlensystems. Dann können wir unsere beiden Zahlen der Länge $2m$ darstellen als

$$x = x_1 B^m + x_0, \quad y = y_1 B^m + y_0.$$

(mit $x_0 < B^m$ und $x_1 < B^m$). Das Produkt ist dann

$$x \cdot y = (x_1 B^m + x_0)(y_1 B^m + y_0) = x_1 y_1 B^{2m} + (x_1 y_0 + x_0 y_1) B^m + x_0 y_0.$$

Und jetzt beobachten wir:

$$\begin{aligned} & x_1 y_0 + x_0 y_1 \\ =& x_1 y_0 + x_0 y_0 + x_0 y_1 + x_1 y_1 - x_1 y_1 - x_0 y_0 \\ =& (x_1 + x_0) y_0 + (x_0 + x_1) y_1 - x_1 y_1 - x_0 y_0 \\ =& (x_1 + x_0)(y_0 + y_1) - x_1 y_1 - x_0 y_0 \end{aligned}$$

**Beispiel 13.1.** Wir berechnen 12345 mal 6789, mit $B = 10$ und $m = 3$. "Mal 1000" zählt hier nicht als Multiplikation, denn wir schieben die Ziffern ja nur um drei Positionen.

$$\begin{aligned} 12345 &= 12 \cdot 1000 + 345 \\ 6789 &= 6 \cdot 1000 + 789 \end{aligned}$$

Wir führen drei (statt vier) Multiplikationen von Zahlen der Länge drei durch:

$z_2 = 12 \cdot 6 = 72$
$z_0 = 345 \cdot 789 = 272205$
$z_1 = (12+345) \cdot (6+789) - z_2 - z_0 = 357 \cdot 795 - 72 - 272205 = 283815 - 72 - 272205 = 11538.$

Wir erhalten das Ergebnis durch Addieren der drei (geeignet verschobenen) Teilergebnisse.

Ergebnis: $72 \cdot 1000^2 + 11538 \cdot 1000 + 272205 = 83810205$.

Für den eigentlichen Karatsuba-Algorithmus wenden wir das nun rekursiv an. Zum Multiplizieren zweier Zahlen der Länge $n = 2^k$ brauchen wir 3 Multiplikationen von Zahlen der Länge $2^{k-1}$, also $3^2$ Multiplikationen von Zahlen der Länge $2^{k-2}$, also ...., also $3^k$ Multiplikationen von Zahlen der Länge eins.

Es ist $k = \log_2(n)$, also brauchen wir

$$3^k = 3^{\log_2(n)} = (2^{\log_2(3)})^{\log_2(n)} = (2^{\log_2(n)})^{\log_2(3)} = n^{\log_2(3)} \approx n^{1,58}$$

Müssen wir uns um die Additionen sorgen? Nein: in jeder Runde sind es $O(n)$ viele. Es gibt $\log_2(n)$ Runden, also $O(n\log(n))$ Additionen insgesamt. Der dominierende Teil ist also das $n^{1,58}$. Also Laufzeit insgesamt $O(n^{1,58})$.

## 13.3 FFT-Algorithmus zur schnellen Multiplikation

### Schnelle Multiplikation von Polynomen

Wollen wir zwei Polynome vom Grad $M-1$ multiplizieren, etwa $p(x) = x^3 + x^2 + x + 1$ und $q(x) = x^3 + x^2 + 1$, dann brauchen wir naiv $M^2$ Multiplikationen (hier $M = 4$):

$$\begin{aligned}(x^3+x^2+x+1)(x^3+x^2+1) &= x^6+x^5+x^4+x^3+x^5+x^4+x^3+x^2+x^3+x^2+x+1\\ &= x^6+2x^5+2x^4+3x^3+2x^2+x+1\end{aligned}$$

(hier ein paar weniger, da ein Koeffizient 0 ist).

Erinnern wir uns an die Definition der Faltung auf Seite 42:

$$f * g(n) = \frac{1}{N}\sum_{k=0}^{N-1} f_k g_{n-k} \qquad (0 \le n \le N-1)$$

Wieder gilt: Der Index $n-k$ von $g_{n-k}$ kann negativ werden, also legen wir hier für $1 \le k \le N-1$ fest: $g_{-k} = g_{N-k}$. Stellen wir die Polynome als Koeffizientenvektoren $p$ und $q$ dar (also $p_0 + p_1 x + p_2 x^2 + \cdots \to (p_0, p_1, p_2, \ldots)$), so ist der Koeffizientenvektor des Produkts der Polynome gegeben durch $M$-mal die Faltung der Vektoren: $Mp * q$. Wegen evtler Überträge müssen wir den Koeffizientenvektor mit $N = 2M$ Koordinaten darstellen. (Also, damit etwa der 0-te Eintrag nur von $f_0$ und $g_0$ abhängt, und nicht auch von $f_1$ und $g_{-1} = g_{M-1}$ etc) Für unser Beispiel also

$$8 \cdot (1,1,1,1,0,0,0,0) * (1,0,1,1,0,0,0,0) = (1,1,2,3,2,2,1,0).$$

Jetzt haben wir folgende tolle Idee:

$$\boxed{\mathrm{N}\, p * q = \mathrm{N}\, IDFT(\widehat{p} \cdot \widehat{q}) = IDFT(\mathrm{N}\widehat{p} \cdot \widehat{q})}$$

In Worten: Die Faltung mal $N$ (also das Produkt der Polynome) berechnet sich nach dem Faltungssatz als inverse DFT des $N$-fachen des Produkts der DFTs von $p$ und $q$. Die DFT und IDFT können wir effizient berechnen. **Obacht:** Was bedeutet das Produkt? Ein Polynom ist gegeben durch seinen Koeffizientenvektor. Jedes Polynom vom Grad $N-1$ ist aber auch eindeutig gegeben durch $N$ Funktionswerte. Die DFT von $p$ ist gerade ($1/N$ mal) der Vektor der Funktionswerte an den (komplexen) Stellen $1, \xi, \xi^2, \ldots \xi^{N-1}$. Also:

$$\widehat{p} = \frac{1}{N}(p(1), p(\xi), p(\xi^2), \ldots, p(\xi^{N-1}))^T$$

Den Koeffizientenvektor von $p$ bekommen wir dann als ($N$ mal) die IDFT von $\widehat{p}$ zurück.

Die Funktionswerte des Produkts $pq$ an den Stellen $1, \xi, \ldots, \xi^{N-1}$ ist der Vektor der Funktionswerte

$$\big(pq(1), pq(\xi), \ldots, pq(\xi^{N-1})\big)^T = \big(p(1)q(1), p(\xi)q(\xi), \ldots, p(\xi^{N-1})q(\xi^{N-1})\big)^T.$$

Den können wir aus $\widehat{p}$ und $\widehat{q}$ einfach berechnen: elementweise multiplizieren. Dessen IDFT liefert dann also die Koeffizientendarstellung von $pq$ ! Nun ist $pq$ ein Polynom von höherem Grad ($2N-2$). Daher brauchen wir mindestens $2N-1$ Funktionswerte. Es bietet sich an, mit Koeffizientenvektoren der Länge $2N$ zu arbeiten. Also:

**Algorithmus** (schnelle Multiplikation nach Hausfrauenart)

Gegeben zwei Polynome $p, q$ vom Grad $N-1$. Seien $p$ und $q$ ihre Koeffizientenvektoren der Länge $2N$. Berechne $\widehat{p}$ und $\widehat{q}$ mittels FFT. Berechne dann die IDFT des Vektors

$$2N(\widehat{p}_0 \cdot \widehat{q}_0, \widehat{p}_1 \cdot \widehat{q}_1, \ldots, \widehat{p}_{2N-1} \cdot \widehat{q}_{2N-1})$$

Ergebnis ist der Koeffzientenvektor von $pq$.

**Beispiel 13.2.** (Etwas anders als das oben) Sei $p(x) = x^3 + x^2 + x + 1$, $q(x) = x^2 + 1$. Also $N = 4$, also arbeiten wir mit Vektoren der Länge 8:

$$p = (1,1,1,1,0,0,0,0), \quad q = (1,0,1,0,0,0,0,0)$$

FFT liefert

$$\widehat{p} = \frac{1}{8}\big(4, 1-(1+\sqrt{2})i, 0, 1+(1+\sqrt{2})i, 0, 1-(1-\sqrt{2})i, 0, 1+(1+\sqrt{2})i\big)^T$$

$$\widehat{q} = \frac{1}{8}\big(2, 1-i, 0, 1+i, 2, 1-i, 0, 1+i\big)^T$$

Damit ist

$$8\widehat{p}\widehat{q} = (1, \frac{1}{8}(1-i)(1-(1+\sqrt{2})), 0, \frac{1}{8}(1+i)(1+(1+\sqrt{2})), 0, \frac{1}{8}(1-i)(1-(1-\sqrt{2})), 0, \frac{1}{8}(1+i)(1+(1+\sqrt{2}))\big)^T$$

und die IDFT dieses Vektors ist $(1,1,2,2,1,1,0,0)^T$. Also ist

$$(x^3 + x^2 + x + 1)(x^2 + 1) = x^5 + x^4 + 2x^3 + 2x^2 + x + 1.$$

**Schnelle Multiplikation großer Zahlen**

Das ist nun einfach: Statt des Koeffizientenvektors eines Polynoms sei der Vektor nun die Binärdarstellung der Länge $2N$ zweier Zahlen mit der Länge $N$. (Man überlege sich: wenn $(p_0, p_1, p_2, \ldots)$ die Binärdarstellung ist, und $p$ das Polynom zu diesem Koeffizientenvektor, was ist dann $p(2)$?) Ein Beispiel:

$$15 = (1,1,1,1,0,0,0,0), \quad 5 = (1,0,1,0,0,0,0,0)$$

Ganz analog wie oben wenden wir den Algorithmus an. Dann ist

$$15 \cdot 5 = (1,1,2,2,1,1,0,0)$$

Das ist so keine Binärzahl, aber Abarbeiten der Überträge (von links nach rechts) liefert die korrekte Binärzahl:

$$(1,1,2,2,1,1,0,0) \to (1,1,0,3,1,1,0,0) \to (1,1,0,1,2,1,0,0) \to (1,1,0,1,0,2,0,0)$$
$$\to (1,1,0,1,0,0,1,0) = 75 = 15 \cdot 5$$

**Bemerkung 13.3.** Das obige Beispiel macht auch klar, warum es besser ist mit Einheitswurzeln in $(Z_N, +, \cdot)$ zu arbeiten statt mit komplexen Einheitswurzeln: Die ersteren sind ganzzahlig und reell (integer), die letzteren weder noch (Brüche, Wurzeln, imaginäre Anteile). Das erste ist numerisch einfacher und stabiler.

Ein Vergleich mit Tabelle auf Seite 51 ergibt: Für Zahlen mit 1024 Bit lohnt sich's schon: Statt (i.Wes.) $1024^2 = 1.048.576$ Operationen für "naive" Multiplikation brauchen wir (i.Wes.) nur drei DFTs der Länge 2048, also weniger als 180.000 Operationen. (Grob grechnet: Für DFT der Länge 1024: 15.360, für eine der Länge 2048 dann wohl weniger als 60.000.)

# References


[AHKKLS] T. Arens, F. Hettlich, Ch. Karpfinger, U. Kockelkorn, K. Lichtenegger, H. Stachel: *Mathematik* (Spektrum Verlag 2008)

[GG] J. von zur Gathen, J. Gerhard: *Modern Computer Algebra*, (Cambridge University Press 1999)

[PLA] R. Plato: *Numerische Mathematik kompakt* (Vieweg und Teubner 2006)

[WIK] Online: `http://en.wikipedia.org`


Kapitel 3 in [PLA] zusammen mit Kapitel 13 in [GG] ist in etwa unser Kapitel 10.