# Zusammenfassung
# AlgoKS

20. Januar 2014

## Inhaltsverzeichnis

# 1 kontinuierliche Daten

Die Menge $\mathbb{R}$ ist überabzählbar unendlich ($\mathbb{N}, \mathbb{Z}$ und $\mathbb{Q}$ dahingegen abzählbar unendlich!)

**Fluch der Dimensionen**
$\mathbb{R}^k \to \mathbb{R}^l$, in den ersten $k$ Stellen eindeutig, $n$ Werte pro Stelle
Speicheraufwand: $l \cdot n^k$ Werte.

# 2 Matrizen

## 2.1 Determinanten

$\det(AB) = \det(A)\det(B) = \det(BA)$

$D$ diagonal $\Rightarrow \det(D) = d_{11}d_{22}...d_{nn}$
$L$ untere Dreiecksmatrix $\Rightarrow \det(L) = l_{11}l_{22}...l_{nn}$
$R$ obere Dreiecksmatrix $\Rightarrow \det(R) = r_{11}r_{22}...r_{nn}$

## 2.2 Singulatität

Matrix $M$ singulär $\Leftrightarrow \det(M) = 0$
Bedeutung: $M$ nicht-singulär $\Rightarrow$ eindeutiges $M^{-1}$ existiert.

## 2.3 Orthogonalität

Matrix $M$ orthogonal $\Leftrightarrow M^{-1} = M^T$
Besteht aus orthonormalen Vektoren (Eigenvektoren)

## 2.4 Eigenwerte

($m \times m$ - Matrix M)
M hat maximal $m$ Eigenwerte (die Nullstellen des char. Polynoms $p(\lambda) = \det(A - \lambda E)$.

## 2.5 Matrixnormen

- $||A||_\infty$: Maximums-Norm
- $||A||_2$: Euklidsche Norm, $||A||_2 = \sigma_1 (= \max \sigma_i$
- $||A||_1$: Summen-Norm
- $||A||_F$: Frobeniusnorm, $||A||_F = \sqrt{\sum_{i=0}^{m} \sum_{j=1}^{n} |a_{ij}^2|}$

# 3 Direkte Verfahren zur Lösung von LGS

## 3.1 Gaußsches Eliminationsverfahren

## 3.2 LR - Zerlegung $\mathcal{O}(n^3)$

**Bestimmen der Zerlegung** $\sim \frac{2}{3}n^3$

```
T = A;
for(int j = 1; j < n; j++) {
  for( int i = j+1; i <= n ; i++) {
    T[i, j] = T[i, j] / T[j, j] ;
    for ( int k = j+1; k = n ; k++) {
      T[i, k] = T[i, k] - T[i, j] * T[j, k];
} } }

for(int j = 1; j <= n; j++) {
  L[j, j] = 1;
  for(int i = 1; i <= n; i++) {
    if(i <= j) R[i, j] = T[i, j];
    else L[i, j] = T[i, j];
} }
```

### 3.2.1 Pivotsuche

$P_{ij}A$: Vertauschen der Zeilen $i$ und $j$
$AP_{ij}$: Vertauschen der Spalten $i$ und $j$

### 3.2.2 Anwendung: Berechnung der Determinante

$\det(A) = \det(L) \cdot \det(R) = 1 \cdot r_{11} \cdot r_{22} \cdot \ldots r_{nn}$

### 3.2.3 Anwendung: Lösen mehrerer Gleichungssysteme mit $A$ als Koeffizientenmatrix

1. Löse $Rx = y$ (Rückwärtseinsetzen, $\mathcal{O}(n^2)$
2. Löse $Ly = b$ (Vorwärtseinsetzen, in $\mathcal{O}(n^2)$)

### 3.2.4 Rechenaufwand

Anhand des Pseudocodes

### 3.2.5 Alternativen?

**Cramer'sche Regel** $x_i = \frac{\det(A_{i,b})}{\det(A)}$, $A_{i,b}$ = Matrix $A$ mit ersetzter Spalte $i$ durch $b$
Berechnen der Determinanten **sehr teuer**, daher hinfällig.

**Inverse Matrix** $x = A^{-1}b$
**Teurer** und **ungenauer** ($\rightarrow$ numerische Stabilität) als die LR-Zerlegung.

### 3.2.6 Spezielle Matrizen

**Bandmatrizen der Ordnung $n$ mit Bandbreite $m$** Aufwand: $\mathcal{O}(m^2 \cdot n)$
Für $m = 3$ gilt:

$$\underbrace{\begin{pmatrix} 1 & 0 & 0 & \dots & 0 & 0 \\ l_1 & 1 & 0 & \dots & 0 & 0 \\ \vdots & & & & & \\ 0 & 0 & 0 & \dots & l_{n-1} & 1 \end{pmatrix}}_{\text{L}}$$

Hier Laufzeit: $\mathcal{O}(n)$ Operationen

**dünn besetzte Matrizen**
Problem „Fill-In“: 0en werden bei der Elemination zerstört, daher wurden viele Algorithmen entwickelt die das verhindern (sollen).

**Cholesky-Zerlegung für pos. def. Matrizen**
Es gilt: $A = A^T$ (symm.)
$x^T Ax \geq 0$ für alle $x \neq 0$

$$A = L \underbrace{D}_{\substack{\text{Diagonalteil von R}}} L^T$$

$$\underbrace{LD^{0.5}}_{\tilde{L}} \underbrace{(D^{0.5})^T L^T}_{\tilde{L}^T}$$

## 3.3 QR - Zerlegung

# 4 Matrixstrukturen

Je nach Problemstellungen sind die Matrizen sehr groß.
Bei $10^7$ Unbekannten: Matrix der Größe $10^7 \times 10^7$: 400 TB bei Datentyp `float`.
**Bsp:** Temperaturverteilung auf einer Platte: $\frac{1}{4}(u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1})$ für $i, j \in \mathbb{N}$

## 4.1 Nur benötigte Werte speichern: $\#non\_zero \cdot 3$

- Wert ($\#non\_zero$)

- Spaltenindex (pro Wert, $\#non\_zero$)
- Zeilenindex (pro Wert, $\#non\_zero$)

### 4.2 Compressed Row Storage (CRS): $\#non\_zero \cdot 2 + \#rows + 1$

- Wert ($\#non\_zero$)
- Spaltenindex (pro Wert, $\#non\_zero$)
- Zeilenpointer (speichern wo sich die Zeile **ändert**, $\#rows + 1$)

Effiziente Matrix-Vektor-Multiplikation

```
for(int i = 0; i<n; i++)
{
  y[i] = 0;
  for(int j = row_ptr[i]; j < row_ptr[i+1]; j++)
  {
    y[i] = y[i] + val[j] * x[col_ind[j]];
  }
}
```

**Nachteile:**

- Einfügen und Löschen schwierig
- kein direkter Random-Access möglich

### 4.3 Block Compressed Row Storage (BCRS)

Dicht besetzte Gebiete werden zu Blöcken zusammengefasst und normal gespeichert, diese werden als Einträge in einer CRS-Matrix verwendet.

**Vorteile:**

- effektiver bei sehr dicht besetzten Gebieten
- leere Blöcke werden nicht gespeichert

### 4.4 Compressed Column Storage (CCS): $\#non\_zero \cdot 2 + \#columnss + 1$

### 4.5 Matrix als Graph

$$a_{ij} = \begin{cases} 1 & \text{falls Kante zwischen } P_i \text{ und } P_j \\ 0 & \text{sonst} \end{cases}$$

Bei unabhängigen Problemen lässt sich das Problem in 2 kleinere Teilprobleme zerlegen. **Beispiel:** Aufteilen des Graphen in 2 gleich große Teilgraphen. Aufwand für Gauß des großen Graphen: $\mathcal{O}(n^3)$
Aufwand für Gauß für die 2 kleineren Graphen: $\mathcal{O}(2\left(\frac{n}{2}\right)^3) = \mathcal{O}(\frac{n^3}{4})$

## 4.6 Blockmatrizen

$$\left(\begin{array}{cc|cc} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ \hline a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \end{array}\right) = \left(\begin{array}{c|c} A_{1,1} & A_{1,2} \\ \hline A_{2,1} & A_{2,2} \end{array}\right)$$

Aufwand für Matrix-Matrix-Multiplikation: $\mathcal{O}(n^{\log_2 7})$ (vs. $\mathcal{O}(n^3)$ bei naiver Multiplikation)

## 4.7 Faltung

| | |
|---|---|
| Kontinuierlich (1D) | $(f * g)(n) = \int f(\tau) \cdot g(t - \tau) \mathrm{d}\tau$ |
| Disket (1D) | $(f * g)(n) = \sum_k f(k) \cdot g(n - k)$ |

**Eigenschaften:**

- $f * g = g * f$ (kommutativ)
- $(f * g) * h = f * (g * h)$ (assoziativ)

Bsp $f(x) = (\dots, -1, 2, 3, 1, 2, 4, -2, 5, \dots)$
$g = \frac{1}{4}(1, 2, 1) = (\dots, 0, 0, \frac{1}{4}, \frac{1}{2}, \frac{1}{4}, 0, \dots)$
$f * g = (\dots TODOkorrektberechnen : D, 2, \underbrace{\frac{5}{4}}_{\frac{1}{4}\cdot 4 + \frac{1}{2}\cdot(-2) + \frac{1}{4}\cdot 5}, \frac{5}{4} \dots$

### 4.7.1 Anwendung: Filter

$$\begin{pmatrix} 1 & 2 & 3 & 2 & 1 \\ 2 & 4 & 6 & 4 & 2 \\ 3 & 6 & 9 & 6 & 3 \\ 2 & 4 & 6 & 4 & 2 \\ 1 & 2 & 3 & 2 & 1 \end{pmatrix}$$ Bartlett-Filter

**Kantendetektion: Sobelfilter**

$$\begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$
Horizontal

$$\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$
Vertikal

## 4.8 Filter

**Separierbare Filter** Aufteilen des 2D-Filters in 2 1D-Filter: Bearbeiten jeder Zeile/-Spalte mit einem 1D-Filter

**Vorteil:** Zweifache Anwendung des 1D-Filters weniger aufwändig.

Filter separierbar $\Leftrightarrow$ Matrix hat Rang 1

**Beispiele**:

$\frac{1}{16}\begin{pmatrix}1&2&1\\2&4&2\\1&2&1\end{pmatrix}$ separierbar

$\frac{1}{5}\begin{pmatrix}0&1&0\\1&1&1\\0&1&0\end{pmatrix}$ Nicht separierbar

# 5 Fourier-Transformation

$$\text{FT: } \tilde{f}(k) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(x) \cdot \exp(-ikx) \mathrm{d}x$$
$$\text{Inverse FT: } \tilde{g}(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(x) \cdot \exp(ikx) \mathrm{d}x$$

Tiefpass: $\chi_a(x) = \begin{cases} 1 & \text{falls } |x| \leq a \\ 0 & \text{sonst} \end{cases}$

$\tilde{\chi}_a(k) = a\sqrt{\frac{2}{\pi}} \cdot \mathrm{sinc}(ak)$; $\mathrm{sinc}(x) = \frac{\sin(x)}{x}$

Dirac-Fkt: $\int_{-\infty}^{\infty} f(x) \cdot \delta_a(x) = f(a)$

$\tilde{\delta}_a(k) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \delta_a(k) \exp(-ikx) \mathrm{d}x = \frac{1}{\sqrt{2\pi}} \exp -ika$

Kamm-Fkt: $C_\tau = \sum_{n \in \mathbb{Z}} \delta_{n\tau}$

$\tilde{C}_\tau = \frac{\sqrt{2\pi}}{\tau} C_{\frac{2\pi}{\tau}}$

Abtasten = Multiplikation mit Kamm

Faltungssatz $\widetilde{f * g} = \sqrt{2\pi} \cdot \tilde{f} \cdot \tilde{g}$ und $\sqrt{2\pi} \cdot \widetilde{f \cdot g} = \tilde{f} * \tilde{g}$

$\widehat{f * g} = \sqrt{2\pi} \cdot \hat{f} \cdot \hat{g}$ und $\sqrt{2\pi} \cdot \widehat{f \cdot g} = \hat{f} * \hat{g}$

## 5.1 Diskretisierung einer Funktion

$$g_h = g(x) \cdot \sum_m \delta(x - r_m) \text{ mit } r_m = mh, m \in \mathbb{Z} \text{ und Maschenweite } h > 0$$
$$\delta(x - r_m) = \delta_{r_m}(x)$$
$$\delta = \delta_0 \text{Dirac ohne Index} := \text{Dirac an 0}$$
$$\delta_{x-a} = \delta_0(x - a) = \delta_a$$

Faltung mit $\delta_a$

$(\delta_a * f)(x) = \int \delta_a(\tau) f(x - \tau) \mathrm{d}\tau = f(x - a)$

Faltung mit Dirac = Rechtsverschiebung von f um a
$(\delta_a + \delta_b) * f(x) = f(x-a) + f(x-b)$

**Abtasttheorem 1.** *Ist das Spektrum $\tilde{g}(k)$ einer kontinulierlichen Funktion g(x)* **bandbegrenzt**, *d.h. $\tilde{g}(k) = 0$ für $|k| \geq k_{max}$,*
*dann kann $g(x)$ aus den Abtastwerten $[g(j \cdot \Delta x)]_{j \in \mathbb{Z}}$* **exakt rekonstruiert** *werden sofern $|\Delta x| \leq \frac{\pi}{k_{max}}$*

**Merkregel**: Abtastrequenz $\frac{1}{\Delta x}$ mind. doppelt so groß wie maximale Frequenz $\frac{k_{\max}}{2\pi}$

### 5.2 Fast-FT - $\mathcal{O}(n \log n)$

$$PA(n) = \begin{pmatrix} A(m) & 0 \\ 0 & A(m) \end{pmatrix} \begin{pmatrix} Id & 0 \\ 0 & M \end{pmatrix} \begin{pmatrix} Id & Id \\ Id & -Id \end{pmatrix}$$

# 6 Quantisierung

## 6.1 Vektorquantisierung

#### LBK-Algorithmus

1. Berechnen der Startlösung
2. Splitting der Codebook-Vektoren:

$$r_i^{(0)} = (1+\beta) r_1^*$$
$$r_{i+L}^{(0)} = (1-\beta) r_1^* a$$

3. Zuordnung der Samples zu Codebook-Vektoren
4. Berechnung der neuen Codebook-Vektoren
5. Falls Fehler größer als Schranke, gehe zu 3
6. Gehe zu 2, falls gewünschte Anzahl der Codebook-Vektoren noch nicht erreicht

# 7 Interpolation

## 7.1 Resampling

- Rekonstruieren der kontinulierlichen Daten
- Resampling der kontinulierlichen Daten
- Umwandeln in diskrete Daten (für zB. Drucker)

## 7.2 Interpolation

Rekonstruktion durch

- stückweise konstante Funktionen
- stückweise lineare Funktionen
- durch (kubisches) Polynom

## 7.3 Lokale Verfahren

Bei lokalen Verfahren hängt der interpolierte Wert nur von von benachbarten Werten ab.

### 7.3.1 nearest neighbor (Fehler: $\mathcal{O}(h)$)

- Suche die nächstegelegene Stützstelle $x_i$.
- Sprungstelle: Beliebig (Zuordnung zu linker/rechter Stützstelle)
- Interpolierter Wert: $y_i$

\+ kein Rechenaufwand
– ungenau (va. bei glattem $f$)
Anw: Treppenfunktion

### 7.3.2 linear (Fehler: $\mathcal{O}(h^2)$)

- Direktes Verbinden der Stützpunkte $x_i$ und $x_{i+1}$ durch Geraden
- Interpolierter Wert: $p(x) = \frac{y_i(x_{i+1}-x)+y_{i+1}(x-x_i)}{x_{i+1}-x_i}$

\+ wenig Rechenaufwand
– ein bisschen ungenau (va. bei glattem $f$)
Anw: Grafikkarten (das am **weitesten verbreitete** Verfahren)

**Berechnen der Stützstellen** $\frac{x-a}{h}$ TODO

### 7.3.3 Catmull-Rom (Fehler: $\mathcal{O}(h^3)$)

- Schätze Ableitung in den Stützpunkten
- Finde auf jedem Teilinverall das eindeutige kubische Polynom $p_i$, welches in beiden Endpunkten die Stützwerte und geschätzten Ableitungen interpoliert[1]:
  $p_i(x) = a_0(x_{i+1} - x)^3 + a_1(x_{i+1} - x)^2(x - x_i) + a_2(x_{i+1} - x)(x - x_i)^2 + a_3(x - x_i)^3$
  mit: $a_0 = \frac{y_i}{(x_{i+1}-x_i)^3}$, $a_1 = 3a_0 + \frac{y_i'}{(x_{i+1}-x_i)^2}$
  Anmerkung: Hermit-Funktion

[1] Formeln **nicht** auswendig für Klausur

**Schätzen der Ableitung** mittels Differenzen:

- Vorwärtsdifferenz: $y'_i = \frac{y_{i+1}-y_i}{x_{i+1}-x_i}$
- Rückwärtsdifferenz: $y'_i = \frac{y_i-y_{i+1}}{x_i-x_{i+1}}$
- Zentrale Differenz (einfach) $y'_i = \frac{y_{i+1}-y_{i-1}}{x_{i+1}-x_{i-1}}$
- Zentrale Differenz (exakt) = Mittel von Vorwärts- und Rückwärtsdifferenz

$$y'_i = \frac{x_i - x_{i-1}}{x_{i+1} - x_{i-1}} y'_{fw} + \frac{x_{i+1} - x_i}{x_{i+1} - x_{i-1}} y'_{bw}$$

## 7.4 Globale Verfahren

Bei globalen Verfahren hängt der interpolierte Wert von allen Werten ab.

### 7.4.1 Polynom

Gesucht $p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} + x^{n-1}$

**Taylorpolynome** Vandermonde-Matrix $\begin{pmatrix} 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{n-1} \\ \dots & \dots & \dots & & x \dots \\ 1 & x_n & x_2^2 & \dots & x_n^{n-1} \end{pmatrix}$

- – voll besetzt
- – nicht singulär
- – schlecht konidioniert
- – aufwändig zu berechnen
- \+ eindeutig lösbar

**Bernsteinpolynome** $B_i^n(x) = \binom{n}{i} (1-x)^{n-1} x^i$

- – nicht geeignet für Polynominterpolation

**Lagrangepolynome** $L_k(x) = \prod_{i=1 i \neq k}^{n} \frac{x-x_i}{x_k-x_i}$

$L_k$ verschwindet an allen Stützstellen außer $x_k$: $L_k(x_i) = \delta_{ik} = \begin{cases} 1 & \text{für i = k} \\ 0 & \text{sonst} \end{cases}$

→ Veringerung des Fehlers durch Tschebychef-Stützstellen

Interpolationsformel: $p(x) = \sum_{k=1}^{n} y_k L_k(x)$

#### Newton-Polynome

$$\begin{aligned} q_0(x) &= 1 \\ q_1(x) &= (x - x_1) \\ q_2(x) &= (x - x_1)(x - x_2) \\ &\vdots \quad \vdots \\ q_n(x) &= \prod_{i=1}^{n} (x - x_i) \end{aligned}$$

Effeziente und stabile Lösung mit **Algorithmus von Aitken-Neville**

Ansatz: $p(x) = a_0 + a_1(x - x_1) + a_2(x - x_1)(x - x_2) + \cdots + a_{n-1}(x - x_1)(x - x_2)\dots(x - x_{n-1})$.

$$\begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 1 & (x - x_1) & 0 & \dots & 0 \\ 1 & (x - x_1) & (x - x_1)(x - x_2) & \dots & 0 \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

Lösen mit: $p_{i,k} = \frac{p_{i+1,k-1} - p_{i,k-1}}{x_{i+k} - x_i}$, $a_i = p_{1,i}$

Fehler $f(x) - p(x) = \frac{1}{(n+1)!} f^{(n+1)}(\xi) \cdot w(x)$

Spezialfall: lin. Interpolation $n = 1$: $f(x) - p(x) = \frac{1}{2!} f^{(2)}(\xi)(x - x_0)(x - x_1)$

### 7.4.2 B-Spline

Eigenschaften: $\#(Freiheitsgrade - \#(Stetigkeitsbediungungen) = m(n-1) - (n-2)(m-1) = mn - m - (nm - 2m - n + 2) = m + n - 2$

# 8 Freiformkurven

Parametrische Berechreibung der Kurven: $C : [a, b] \to \mathbb{R}^d$

## 8.1 Bernstein-Polynome

$B_i^n = \binom{n}{i} (1 - t)^{n-i} t^i$

$B = \text{span}\{B_0^n, \dots, B_n^n\}$.

Eigenschaften von $B_i^n$:

- $0 \leq B_i^n \leq 1$
- $B_i^n$ hat $i$-fache Nullstelle in $t = 0$
- $B_i^n$ hat $n - i$-fache Nullstelle in $t = 1$
- $\sum_{i=0}^{n} B_i^n(t) = 1$, $t \in [0, 1]$

## 8.2 Bezierkurven

Gegeben seien **Kontrollpunkte** $b_0, \dots b_n \in R^d$.
Sie beschreiben eine Bezierkurve im $\mathbb{R}^d$: $C(t) = \sum_{i=0}^{n} b_i B_i^n(t)$, $t \in [0,1]$

- **Interpolation der Endpunkte**
  $C(0) = \sum_{i=0}^{n} b_i \cdot B_i(0) = b_0$
  $C(1) = \sum_{i=0}^{n} b_i \cdot B_i(1) = b_n$
- **In den Endpunkten tangential an das Kontroll-Polygon** $C'(t) = \sum_{i=0}^{n} b_i \cdot B_i'(t)$
- **Bezierkurve liegt in der konvexen Hülle**
  M ist konvex $\Leftrightarrow a, b \in M, 0 < t < 1 : (1-t)a + tb \in M$
- **affine Invarianz**
  Affine Abbildung: Tanslation und lineare Abbildung (zB. Rotation, Skalierung, Scherung....)
  $\Phi(C(t)) = \sum_{i=0}^{n} \Phi(b_i) \cdot B_i^n(t)$
- **Variationsreduzierend**
  Für beliebige Grade $g$ gilt:
  # Schnittpunkte $g$ mit Kurve $\leq$ # Schnittpunkte $g$ mit Kontrollpolygon

#### Algorithmus von de Casteljau

## 8.3 Bilineare Interpolation

Durch zweifache lineare Interpolation: $(1-s)(1-t)P_1 + s(1-t)P_2 + (1-s)tP_3 + stP_4$

- **Interpolation der Endpunkte**
- **In den Endpunkten tangential an das Kontroll-Polygon**
- **Bezierkurve liegt in der konvexen Hülle**
- **affine Invarianz**
- ~~**Variationsreduzierend**~~

## 8.4 Coons-Patch

$$F(s,t) = \underbrace{F_t(s,t)}_{\text{linearer Blend von } C_w \text{ und } C_o} + \underbrace{F_s(s,t)}_{\text{linearer Blend von } C_s \text{ und } C_n} - \underbrace{F_{st}(s,t)}_{\text{bilinearer Interpoland}}$$

$$F_s(s,t) = (1-s)C_w(t) + sC_o(t)$$
$$F_t(s,t) = (1-t)C_s(s) + tC_n(s)$$
$$F_{st}(s,t) = (1-s)(1-t)C_w(0) + (1-s)tC_w(1) + s(1-t)C_o(0) + stC_o(1)$$

# 9 Interpolation multivariater Daten

Euler-Formel für planares Gitter ohne Löcher: $|F| - |E| + |V| = 1$

**Speicherstruktur: shared vertex / indexed face set** besteht aus:

- Liste der Punkte (vertex list, enthält die Koordinaten der Ecken)
- Liste der Zellen (face list, enthält Indizes der beteiligten Vertexes)

## 9.1 Lineare Interpolation

### 9.1.1 Baryzentrische Koordinaten

$P = \rho R + \sigma S + \tau + T,\ \rho + \sigma + \tau = 1$

$$
\begin{aligned}
P &= \begin{pmatrix} x \\ y \end{pmatrix} \\
R &= (R_x, R_y) \\
S &= (S_x, S_y) \\
T &= (T_x, T_y) \\
x &= \rho R_x + \sigma S_x + \tau + T_x \\
y &= \rho R_y + \sigma S_y + \tau + T_y \\
1 &= \rho + \sigma + \tau
\end{aligned}
$$

### 9.1.2 Globale Verfahren

**Tenorproduktansatz** Polynominterpolation nicht möglich, da die Zahl der Freiheitsgrade problematisch ist.
Ggf. mehrdeutig oder unlösbar.

**Radiale Basisfunktion (RBF)** $\tilde{h}$

$$\tilde{h} =$$

# 10 Singulärwertzerlegung

$A = U\Sigma V^T$
Die Spalten von $U$ enthalten die (orthogonalen) EVs von $AA^T$.
$\Sigma$ ist Diagonalmatrix, $\Sigma = \text{diag}(\sigma_1, ..., \sigma_r)$, wobei $\sigma_i$ Singulärwert von $AA^T$, $\sigma_i = \sqrt{\lambda_i}$.
Die Spalten von $V$ enthalten die (orthogonalen) EVs von $A^T A$.

### 10.1 Anwendung: low-rank-approximation

Ziel: Finde Matrix $A_k$ mit Rang $k$, so dass gilt: $A_k = \min_{X:rank(X)=k} ||A - X||_F$.

Dies erreicht man mithilfe der SVD: $A_k = U\text{diag}(\sigma_1, ...\sigma_k, \underbrace{0..., 0}_{\text{r-k}})V^T$

Es gilt: $||A - A_k||_F = \sigma_{k+1}$ (Annahme: Singulärwerte nach der Größe sortiert).

### 10.2 Anwendung: Lösen singulärer Gleichungen

Pseudo-Inverse von $A$: $A^{\sim 1} = V\Sigma^{\sim 1}U^T$, wobei $\Sigma^{\sim 1} = \text{diag}(\frac{1}{\sigma_1}, ..., \frac{1}{\sigma_m})$

Ziel: Löse $A\vec{x} = \vec{b}$.

$\vec{x} = A^{\sim 1}\vec{b} = \sum_{i=1}^{r} \frac{1}{\sigma_i}(u_i^T b)v_i$. Für die Lösung gilt: $||Ax - b||_2 \to \min$, $||x||_2 \to \min$

### 10.3 Numerische Bestimmung der SVD

1. Transformation auf Tridiagonalgestalt
   a) Ähnlichkeitstransformation
   $Q_i = Id - 2n_i n_i^T$ $Q_1 A Q_1 =) ($
2. EWs und EVs der Tridiagonalmatrix bestimmen
   a) QR-Verfahren $A_0 = Q_0 R_0 \to A_1 := R_0 Q_0$

# 11 Hauptachsentransformation - PCA

### 11.1 Bestimmung der PCA

1. Balancieren der Daten
   Berechne den Mittelwert: $\overline{M} = \frac{1}{N}\sum_i X_i$
   Balancierte Daten: $\overline{X}_i = X_i - \overline{X}$
2. Bilde die Kovarianzmatrix
   $C = \text{cov}(X, X) = \frac{1}{N-1}\sum_i \overline{X}_i \overline{X}_i^T$
3. Diagonalisieren der Kovanianzmatrix
   $$C = Q \begin{pmatrix} \lambda_1 & 0 & ... & 0 \\ 0 & \lambda_2 & & \\ \vdots & & & \vdots \\ 0 & & ... & \lambda_n \end{pmatrix} Q^{-1}.$$
   Die Spalten von $Q$ sind die Eigenvektoren (bezeichnet als „Hauptachsen“).

### 11.2 QR-Zerlegung

$A = QR$, wobei $Q$ orthogonal, $R$ Dreiecksmatrix.

$+$ Keine Pivotisierung notwendig
$+$ $\kappa_2(R) = \kappa_2(A)$
$+$ $\kappa_2(Q) = 1$
$\rightarrow$ QS stabiler und besser konditioniert

### 11.2.1 Jacobi-Rotationen $\mathcal{O}(n^3)$

Einfügen von 0en in einer Zelle (unter der Diagonalen).

$$\begin{pmatrix} 1 & & & & & & & & \\ & \ldots & & 0 & & 0 & & & 0 \\ & & 1 & & & & & & \\ & & & c & & & -s & & \\ & & & & 1 & & & & \\ & 0 & & & & \ldots & & & 0 \\ & & & & & & 1 & & \\ & & & s & & & & c & \\ & & & & & & & & 1 \\ & 0 & & 0 & & 0 & & \ldots & \\ & & & & & & & & & 1 \end{pmatrix}$$

$J_{ij}(\phi)^{-1} = J_{ij}(-\phi) = J_{ij}(\phi)^T$

### 11.2.2 Householder-Spiegelungen $\mathcal{O}(n^3)$

Einfügen von 0en in einer Spalte (unter der Diagonalen).
$H_w = Id - \frac{2}{w^T \cdot w} \cdot w \cdot w^T$
$H_w x = x - \frac{2 \cdot w^T \cdot x}{w^T \cdot w} \cdot w$
$H_w^{-1} = H_w^T = H_w$

## 11.3 QR-Verfahren

Auf QR-Zerlegung basierendes iteratives Verfahren zur Bestimmung der Eigenwerte.

$A_1$ bestimme QR-Zerlegung $A = QR$ und setze $A_1 = RQ$(...

## 11.4 Spezialfall QR-Zerlegung für tridiagonale Matrizen

QR-Zerlegung mit nur $n-1$ Jacobi-Rotationen, da pro Spalte nur ein Eintrag unterhalb der Diagonalen $\neq 0$ ist.

$$\underbrace{H_1}_{\text{Householder}} \quad A = \begin{pmatrix} * & * & \ldots & * \\ * & * & & \\ & & & \\ 0 & * & & \end{pmatrix}$$

### 11.5 Bestimmung der Ausgleichsgerade mit Normalengleichungen

Lösen der Gleichung $A^T Ax = A^T b$

$$A^T Ax = \begin{pmatrix} 1 & \dots & 1 \\ x_1 & \dots & x_2 \end{pmatrix} \cdot \begin{pmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix} = \begin{pmatrix} n & \sum x_i \\ \sum x_i & \sum x_i^2 \end{pmatrix}$$

$$A^T b = \begin{pmatrix} \sum y_i \\ \sum x_i y_i \end{pmatrix}$$

# 12 Iterative Verfahren

## 12.1 Fixpunktiteration

Sei $\phi : M \to M$ Selbstabbildung.
$M$ Fixpunkt, wenn gilt: $\phi(M) = M$

Wenn $x_{x+1} = \phi(x_i)$ konvergiert, $x^* = \lim\limits_{i \to \infty} x_i$ dann gilt: $\phi(x^*) = x^*$

### 12.1.1 Banachscher Fixpunktsatz

Sei $I$ ein abgeschlossenes Intervall und $\phi : I \to I$ eine Kontraktion, dh. es gibt eine Konstante $L < 1$, so dass

$$|\phi(x) - \phi(y)| \leq L|x - y| \text{ für alle } x, y \in I$$

dann gilt:

- $\phi$ besitztgenau einen Fixpunkt $x^* \in I$
- die Iterationsfolge $x_{i+1} = \phi(x_i)$ konvergiert für jeden Startwert $x_0 \in I$

## 12.2 Nullstellenbestimmung: Newton-Verfahren

Starte mit gegebenem Startwert $x_0$
Bilde die Tangente im Punkt $(x_0/f(x_0))$ und bestimme ihre Tangente
Nullstelle: $x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$

**Konvergenz:** Newton-Verfahren konvergiert, falls $x_0$ nahe der Nullstelle.

### 12.2.1 im $R^n$

$x_{i+1} = x_i - [J_F(x_i)]^{-1} F(x_i)$

## 12.3 Nullstellenbestimmung: Sekanten-Verfahren

Starte mit 2 gegebenen Startwerten $x_0$und $_1$ mit unterschiedlichen Vorzeichen.
Bestimme den Schnittpunkt der Sekante druch $(x_0/f(x_0)$ und $(x_1/f(x_1)$

$x_{i+1} = \frac{x_{i-1}f(x_i) - x_i f(x_{i-1}}{f(x_i) - f(x_{i-1})}$

## 12.4 Nullstellenbestimmung: Bisektionsverfahren

Starte mit 2 gegebenen Startwerte $x_0$ und $x_1$, $x_0 < x_1$ mit unterschiedlichen Vorzeichen.
Bestimme den Mittelpunkt $x_2 = (x_0 + x_1)/2$ und betrachte das Interval, dessen Grenzen unterschiedliche Vorzeichen hat.

## 12.5 Nullstellenbestimmung: Regula Falsi

Erweiterung des Bisektionsverfahrens: **TODO**

## 12.6 Bestimmung von $\sqrt{a}$: Heron-Verfahren

$x_{i+1} = \frac{x_i + \frac{a}{x_i}}{2}$ A

## 12.7 Lösen von LGS: Jacobi-Iteration $V_{iter} = -D^{-1}(L+R)$, $A_0 = D$

Zerlegung der Matrix $A = L + D + R$
$Dx^{i+1} + (L+R)x^i = b$
$x^{i+1} = D^{-1}(b - Lx - Rx)$

```
for(i = 1; i <= maxIter; i++) // Anzahl Iterationen
  for(k = 1; k <= n; k++) // ein Schritt
  {
    x_k^{i+1} = (b_k - sum_{j=1...k-1} a_{k...j} x_j^i - sum_{j=k+1...n} a_{k...x_j^i})/a_kk
  }
```

## 12.8 Lösen von LGS: Gauss-Seidel-Iteration

Wie Jacobi, nur werden bereits berechnete Werte direkt verwendet.
Zerlegung der Matrix $A = L + D + R$
$(L+D)x^{i+1} + Rx^i = b$

```
for(i = 1; i <= maxIter; i++) // Anzahl Iterationen
  for(k = 1; k <= n; k++) // ein Schritt
  {
    x_k^{i+1} = (b_k - sum_{j=1...k-1} a_{k...j} x_j^{i+1} - sum_{j=k+1...n} a_{k...x_j^i})/a_kk
  }
```

## 12.9 Lösen von LGS: SOR-Iteration

```
for(i = 1; i <= maxIter; i++) // Anzahl Iterationen
  for(k = 1; k <= n; k++) // ein Schritt
  {
    x_k^{i+1} = x_k^i + ω((b_k − Σ_{j=1...k−1} a_{k,j} x_j^{i+1} − Σ_{j=k+1...n} a_{k,j} x_j^i)/a_kk − x_k^i)
  }
```

Oder einfacher: $x_k^{i+1} = (1-\omega)x_k^i + \omega(b_k - \sum_{j=1\ldots k-1} a_{k,j}x_j^{\mathbf{i+1}} - \sum_{j=k+1\ldots n} a_{k,j}x_j^i)/a_{kk}$
Für alle 3 Verfahren gilt:

- lineare Konvergenz (je größer die Systemmatrix, desto langsamer)
- SOR konvergiert, wenn $A$
- GS / J konvergieren, wenn:
  - $A$ strikt **diagonaldominant**

$$|a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}| \quad \forall i$$

  - $A$ **unzerlegbar** und **schwach diagonaldominant** Unzerlegbarkeit: Matrix $A$ als Graph interpretieren.

$$|a_{ii}| \geq \sum_{j=1, j\neq i}^{n} |a_{ij}| \quad \forall i$$

    wobei für ein $i >$ gelten muss!
- In vielen Fällen braucht GS nur halb so viele Schritte wie J

## 12.10 cg-Verfahren

$F(x) = x^T Ax + 2b^T x + y$
Vektoren $u$ und $v$ A-konjugiert, falls gilt: $u^T Av = 0$
Zur Lösung von LGS: $\mathcal{O}(n^3)$, für dünn besetzte Matrix $\mathcal{O}(n^2)$

## 12.11 Quasi-Newton-Verfahren

$(x^{(k)} - x^{(k-1)} = -H_k(grad(F)(x^{(k)}) - grad(F)(x^{(k-1)})$