TN
Teilnehmernummer
(von Platzkarte)

Prof. Dr. Günther Greiner
Lehrstuhl für Graphische Datenverarbeitung
der Universität Erlangen–Nürnberg

24. Juli 2015

Algorithmik kontinuierlicher Systeme — 24. Juli 2015

Angaben zur Person (Bitte in DRUCKSCHRIFT ausfüllen!):

Name, Vorname: .....

Geburtsdatum: .....

Matrikelnummer: .....

Studienfach: .....

Nicht von der Kandidatin bzw. vom Kandidaten auszufüllen !

Bewertung:

Aufgabe	1	2	3	4	5	6	7	8	9	10
Max. Punktzahl	6	8	10	13	9	9	7	11	8	9
Erreichte Punkte										

Gesamtpunktzahl	
Note	

Organisatorische Hinweise

Die folgenden Hinweise bitte aufmerksam lesen und die Kenntnisnahme durch Unterschrift bestätigen!

- Bitte legen Sie Ihren Studentenausweis und einen Lichtbildausweis zur Personenkontrolle bereit.
- Hilfsmittel (außer Schreibmaterial und Taschenrechner) sind nicht zugelassen. Andere elektronische Geräte sind auszuschalten.
- Fragen zu den Prüfungsaufgaben werden grundsätzlich nicht beantwortet.
- Die Lösung einer Aufgabe muss auf das jeweilige Aufgabenblatt geschrieben werden. Sollte der Platz nicht reichen, so verwenden Sie die Zusatz-Seiten am Ende der Klausur. Fügen Sie einen Hinweis in Ihre Lösung ein, dass die Lösung auf den Zusatz-Seiten fortgesetzt wurde und beschriften Sie diese mit Namen und Aufgabennummer.
- Es können durch die Aufsicht zusätzlich Seiten eingeheftet werden, sollte mehr Platz benötigt werden. Bitte beschriften Sie den Kopf dieser Seiten mit Ihrem Namen und der Aufgabennummer. Streichen Sie alles, was nicht bewertet werden soll, doppelt aus.
- Auf Ihrem Platz befinden sich einige lose Blätter Schmierpapier. Bei Bedarf können Sie zusätzliches Schmierpapier von der Aufsicht anfordern. Das Schmierpapier muss abgegeben werden, es wird aber nicht bewertet.
- Wenn Sie die Prüfung aus gesundheitlichen Gründen abbrechen müssen, so muss Ihre Prüfungsunfähigkeit durch eine Untersuchung bei einem Vertrauensarzt nachgewiesen werden. Melden Sie sich bei der Aufsicht und lassen Sie sich das entsprechende Formular aushändigen.
- Die Bearbeitungszeit beträgt 90 Minuten.
- Überprüfen Sie die Prüfungsaufgaben auf Vollständigkeit (30 Seiten inklusive Deckblatt) und einwandfreies Druckbild.
- Vergessen Sie nicht, auf dem Deckblatt die Angaben zur Person inklusive der Teilnehmernummer (TN) von Ihrer Platzkarte einzutragen und die **Erklärungen auf dieser Seite zu unterschreiben**.
- Viel Erfolg!

Erklärungen

Durch meine Unterschrift bestätige ich den Empfang der vollständigen Klausurunterlagen und die Kenntnisnahme der obigen Informationen.

Erlangen, 24. Juli 2015 .....

(Unterschrift)

1 Theoriefragen (6 Punkte)

a) Beantworten Sie die folgenden Fragen! Schreiben Sie ihre Antwort in die rechte Spalte der Tabelle!

Welche Komplexität hat die Matrix-Vektor-Multiplikation $\mathbf{A}\vec{b}$, wenn $\mathbf{A}$ eine tridiagonale $(n \times n)$-Matrix ist und $\vec{b}$ ein n-Vektor ist?	$\mathcal{O}($ $)$
Welche Komplexität hat die Berechnung der EUKLIDISCHEN Norm eines n-Vektors?	$\mathcal{O}($ $)$
Welche Komplexität hat das Lösen eines Gleichungssystems $\mathbf{A}\vec{x} = \vec{b}$, wenn $\mathbf{A}$ eine $(n \times n)$-Matrix und $\vec{b}$ ein n-Vektor ist und die LR-Zerlegung $\mathbf{A} = \mathbf{LR}$ gegeben ist?	$\mathcal{O}($ $)$
Welche Komplexität hat die Berechnung der QR-Zerlegung $\mathbf{A} = \mathbf{QR}$ einer $(n \times n)$-Matrix $\mathbf{A}$?	$\mathcal{O}($ $)$
Welche Komplexität hat die Durchführung eines Iterationsschrittes des JACOBI-Verfahrens für eine vollbesetzte $(n \times n)$-Matrix?	$\mathcal{O}($ $)$
Welchen Grad hat eine BÉZIER-Kurve, die durch n Kontrollpunkte definiert ist?	
Wie groß ist der Approximationsfehler des CATMULL-ROM-Interpolanten im Falle äquidistanter Schrittweite h ?	$\mathcal{O}($ $)$
Wie groß ist der Approximationsfehler der iterierten SIMPSON-Regel für die Schrittweite h ?	$\mathcal{O}($ $)$

b) Nennen Sie ein Iterationsverfahren das quadratisch konvergiert.

c) Was bedeutet **quadratische Konvergenz** (bzw. Konvergenzordung 2)?

2 Direkte Verfahren für lineare Gleichungssysteme (8 Punkte)

a) Bestimmen Sie die LR-Zerlegung folgender Matrix mit Zeilen-Pivotisierung.

$$\mathbf{A} = \begin{bmatrix} 0 & 4 & 2 \\ 4 & 6 & 4 \\ -2 & -1 & 3 \end{bmatrix}$$

$$= \underbrace{\begin{bmatrix} & & \\ & & \\ & & \end{bmatrix}}_{P_0} \cdot \underbrace{\begin{bmatrix} & & \\ & & \\ & & \end{bmatrix}}_{A^*}$$

$$= P_0 \cdot \underbrace{\begin{bmatrix} & & \\ & & \\ & & \end{bmatrix}}_{L^*} \cdot \underbrace{\begin{bmatrix} & & \\ & & \\ & & \end{bmatrix}}_{A^{**}}$$

$$= P_0 \cdot \underbrace{\begin{bmatrix} & & \\ & & \\ & & \end{bmatrix}}_{L} \cdot \underbrace{\begin{bmatrix} & & \\ & & \\ & & \end{bmatrix}}_{R}$$

b) Von der Matrix $\mathbf{C}$ ist die LR-Zerlegung bekannt, es gilt

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

Lösen Sie das folgende lineare Gleichungssystem (unter Verwendung der LR-Zerlegung)

$$\mathbf{C}\vec{x} = \vec{b} \qquad \text{für} \qquad \vec{b} = [0, 1, -4, 2]^T.$$

c) Bei der Übertragung einer QR-Zerlegung der Matrix $\mathbf{B}$ sind Daten verloren gegangen. Bestimmen Sie die fehlenden Einträge, sodass gilt:

$$\mathbf{B} = \begin{bmatrix} -1 & 1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & 1 & 0 & 2 \\ 1 & -1 & -2 & 0 \end{bmatrix} = \underbrace{\frac{1}{2} \cdot \begin{bmatrix} & 1 & -1 & 1 \\ 1 & & 1 & -1 \\ 1 & 1 & & 1 \\ 1 & & & 1 \end{bmatrix}}_{Q} \cdot \underbrace{\begin{bmatrix} & 0 & -2 & 1 \\ 0 & & & 0 \\ 0 & 0 & 1 & 2 \\ & 0 & 0 & 1 \end{bmatrix}}_{R}$$

3 Singulärwertzerlegung (SVD) (10 Punkte)

Die Singulärwertzerlegung einer Matrix $\mathbf{A}$ hat bekanntermaßen die Darstellung $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$.

a) Geben Sie für den zweidimensionalen Fall ($\mathbf{A} \in \mathbb{R}^{2\times 2}$) eine geometrische Interpretation für die Singulärwertzerlegung an. D.h., wie wird ein Vektor $\vec{x} \in \mathbb{R}^2$ durch die Matrizen $\mathbf{V}^T$, $\mathbf{\Sigma}$ und $\mathbf{U}$ transformiert, wenn man $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T\vec{x}$ berechnet?

- Wirkung von $\vec{x} \mapsto \mathbf{V}^T\vec{x}$:

- Wirkung von $\vec{y} \mapsto \mathbf{\Sigma}\vec{y}$:

- Wirkung von $\vec{z} \mapsto \mathbf{U}\vec{z}$:

b) In welchem Zusammenhang stehen die Eigenwertzerlegungen von $\mathbf{A}^T\mathbf{A}$ bzw. $\mathbf{A}\mathbf{A}^T$ und die Sinuglärwertzerlegung von $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$?

Tipp: Eigenwertzerlegungen haben immer die Form $\mathbf{E}\mathbf{D}\mathbf{E}^T$, wobei die Spalten von $\mathbf{E}$ die Eigenvektoren und die Einträge der Diagonalmatrix $\mathbf{D}$ die Eigenwerte sind.

Betrachten Sie nun die Matrix $\mathbf{A} = \frac{1}{25}\begin{bmatrix} 104 & -32 & -56 & -32 \\ 32 & 4 & 32 & 44 \\ -122 & 76 & 8 & 76 \\ 26 & -28 & 26 & 67 \end{bmatrix}$, deren Singulärwertzerlegung bekannt ist:

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \underbrace{\frac{1}{5}\cdot\begin{bmatrix} 3 & 0 & -4 & 0 \\ 0 & -3 & 0 & -4 \\ 4 & 0 & -3 & 0 \\ 0 & -4 & 0 & 3 \end{bmatrix}}_{\mathbf{U}} \underbrace{\begin{bmatrix} 8 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\mathbf{\Sigma}} \underbrace{\frac{1}{5}\cdot\begin{bmatrix} 4 & -2 & -1 & -2 \\ -2 & 1 & -2 & -4 \\ -1 & -2 & 4 & -2 \\ -2 & -4 & -2 & 1 \end{bmatrix}}_{\mathbf{V}^T}$$

c) Bestimmen Sie die Kondition $\kappa_2(\mathbf{A})$ der Matrix $\mathbf{A}$ bezüglich der 2-Norm.

d) Bestimmen Sie das Bild $\mathrm{im}(\mathbf{A})$ und den Kern $\ker(\mathbf{A})$ der Matrix $\mathbf{A}$.

$\mathrm{im}(\mathbf{A}) \quad =$

$\ker(\mathbf{A}) \quad =$

e) Bestimmen Sie die beste Rang-1-Approximation von $\mathbf{A}$ (im Sinne der Frobenius-Norm).

4 Programmierung: Median Cut (13 Punkte)

In dieser Aufgabe soll der Algorithmus *Median Cut* in `C++` implementiert werden. Dazu wird die Klasse `MedianCut` verwendet. Im Gegensatz zu den Übungsaufgaben ist **keine** Fehlerbehandlung erforderlich.

Tipp: Sie können die Teilaufgaben a) bis c) auch ohne Wissen über den Median-Cut-Algorithmus bearbeiten.

Der Algorithmus wird dazu verwendet, eine Punktewolke auf wenige Repräsentanten zu reduzieren.

Die folgende Abbildung veranschaulicht den aus der Vorlesung bekannten Ablauf des Algorithmus:

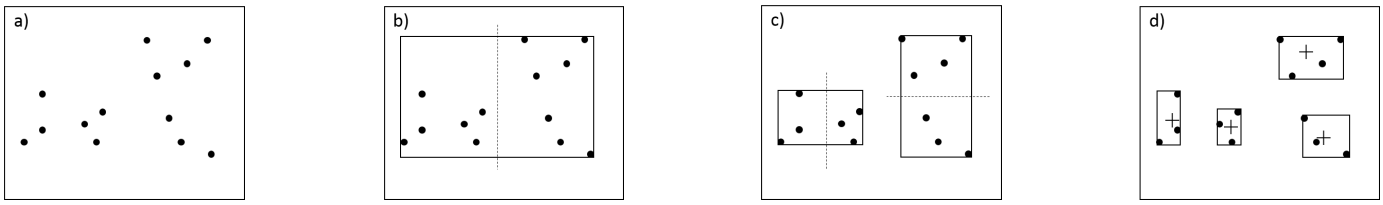


Abbildung 1: Ablauf des Median Cut Algorithmus

Entgegen der Vorlesung werden in dieser Aufgabe die Repräsentanten (Kreuze) nicht als Mittelpunkt der Bounding Box, sondern als Mittelwert der entsprechenden Punkte gewählt.

Hinweis: Sie finden die Klassenstrukturen auch auf der Seite 29, die zum Heraustrennen vorgesehen ist.

Die Klasse `MedianCut` hat die folgende Struktur:

```
class MedianCut {
public:
 //! Klassen Konstruktor
 MedianCut(const std::vector<Point2d>& pointcloud);

 //! Einstiegspunkt zur Berechnung des Median Cut
 std::vector<Point2d> ComputeMedianCut(int nCuts);

private:
 //! Berechne Bounding Box (min, max) der übergebenen Punktewolke pc
 static void ComputeBoundingBox(const std::vector<Point2d>& pc,
                                Point2d& min, Point2d& max);

 //! Sortiert die übergebene Punktewolke entlang der X- bzw. Y-Achse
 static void SortPointCloudAlongXAxis(std::vector<Point2d>& pointcloud);
 static void SortPointCloudAlongYAxis(std::vector<Point2d>& pointcloud);

 //! Gibt den Repräsentanten der Punkte zurück (hier: Mittelwert)
 static Point2d GetRepresentative(const std::vector<Point2d>& pointcloud);

 //! Rekursive Berechnung des Median Cut
 //! (nCuts: maximale Anzahl der Schritte, cCuts: aktueller Schritt)
 static void MedianCutRecursion(std::vector<Point2d> subpointcloud,
                          int cCuts, int nCuts, std::vector<Point2d>& result);

 //! Klassenvariable
 std::vector<Point2d> m_pointcloud;
};
```

Hinweis: Verändern Sie die Klassenstruktur nicht, d.h. führen Sie keine neuen Attribute oder Methoden ein.

Der zu implementierende Algorithmus erhält als Eingabe einen `std::vector<Point2d>`, der eine 2D-Punktewolke repräsentiert. Ein `Point2d` hat folgende – aus den Übungen bekannte – Struktur:

```
class Point2d {
public:
  float x, y;                    //! Interne Darstellung
  Point2d(float x, float y);     //! Konstruktor
  ...                            //! weitere Konstruktoren und Operatoren
};
```

Hinweis: Sie können davon ausgehen, dass für Objekte der Klasse `Point2d` alle arithmetischen Standardoperatoren (`+, -, *, /, =, ==, !=, +=, -=, *=, /=`) zur Verfügung stehen.

a) Implementieren Sie zunächst den Konstruktor der `MedianCut`-Klasse, der als Parameter eine Referenz auf eine konstante 2D-Punktewolke besitzt und diese in die Membervariable `m_pointcloud` abspeichert.

b) Implementieren Sie die Funktion `void ComputeBoundingBox(const std::vector<Point2d>& pc, Point2d& min, Point2d& max)`, welche die Bounding Box (`min`, `max`) der übergebenen Punktewolke `pc` berechnet.

Hinweis: Sie können die Anzahl der Elemente in einem `std::vector` mit der Methode `size()` abrufen. Außerdem dürfen Sie davon ausgehen, dass die Punktewolke immer mindestens ein Element enthält.

```
void MedianCut::ComputeBoundingBox(const std::vector<Point2d>& pc,
                                   Point2d& min, Point2d& max) {

}
```

c) Implementieren Sie die Funktion `Point2d GetRepresentative(const std::vector<Point2d>& pointcloud)`, welche den Repräsentanten für die übergebenen Punktewolke liefert. Dazu soll der Mittelwert der Punkte berechnet und zurückgegeben werden.

Hinweis: Sie können die Anzahl der Elemente in einem `std::vector` mit der Methode `size()` abrufen. Außerdem dürfen Sie davon ausgehen, dass die Punktewolke immer mindestens ein Element enthält.

```
Point2d MedianCut::GetRepresentative(const std::vector<Point2d>& pointcloud) {
    Point2d result;









    return result;
}
```

d) Implementieren Sie die Methode `std::vector<Point2d> ComputeMedianCut(int nCuts)`. Diese ist der Einstiegspunkt der rekursiven Berechnung des Median Cut Algorithmus und ruft intern die Methode `void MedianCutRecursion(...)` mit den entsprechenden initialen Startwerten auf. `nCuts` gibt dabei die Anzahl der auszuführenden Cut-Schritte an (d.h. maximale Rekursionstiefe).

```
std::vector<Point2d> MedianCut::ComputeMedianCut(int nCuts) {
    std::vector<Point2d> result;









    return result;
}
```

e) Die Methode `MedianCutRecursion(std::vector<Point2d> subpointcloud, int cCuts, int nCuts, std::vector<Point2d>& result)` berechnet rekursiv den Median Cut. Dazu wird nach Bestimmung der Bounding Box die Punktewolke entweder entlang der X- oder entlang der Y-Achse sortiert und anschließend die Methode rekursiv mit der halbierten Punktewolke aufgerufen. Im Basisfall wird mit der Methode `GetRepresentative(...)` ein Repräsentant berechnet und an den Ergebnisvektor `result` angehängt (z.B. mittels `push_back(...)`).

Hinweis: Benutzen Sie zum Sortieren der Punktewolke entlang der X-Achse bzw. Y-Achse die Methoden `SortPointCloudAlongXAxis(...)` bzw. `SortPointCloudAlongYAxis(...)`, die in der Klasse `MedianCut` bereitgestellt werden. Sie dürfen für die Klasse `std::vector` alle bekannten Methoden verwenden.

```
void MedianCut::MedianCutRecursion(std::vector<Point2d> subpointcloud, int cCuts,
                        int nCuts, std::vector<Point2d>& result) {

    if (subpointcloud.empty()) return;









}
```

5 Iterative Lösungsverfahren (9 Punkte)

Gegeben seien die 4×4-Matrix $\mathbf{A}$ sowie der Vektor $\vec{b}$ mit

$$\mathbf{A} = \begin{bmatrix} 2 & 0 & 1 & -1 \\ 1 & 2 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 2 & 2 & 0 & 2 \end{bmatrix} \qquad \vec{b} = \begin{bmatrix} 4 \\ 2 \\ 2 \\ 4 \end{bmatrix} .$$

a) Führen Sie **einen** Schritt des JACOBI-Verfahrens zur Lösung von $\mathbf{A}\vec{x} = \vec{b}$ durch. Verwenden Sie als Startvektor $\vec{x}^0 = [1, -1, 1, -1]^T$.

b) Führen Sie **einen** Schritt des **SOR**-Verfahrens mit Relaxationsparameter $\omega = \frac{3}{2}$ zur Lösung von $\mathbf{A}\vec{x} = \vec{b}$ durch. Verwenden Sie als Startvektor $\vec{x}^0 = [1, -1, 1, -1]^T$.

c) Die Lösung der diskretisierten Poissongleichung

$$4u_{i,j} - u_{i+1,j} - u_{i,j+1} - u_{i-1,j} - u_{i,j-1} = 0$$

auf einen äquidistanten, rechteckigen Gitter mit Gitterabstand h wird iterativ mit dem JACOBI-Verfahren gelöst.

In der folgenden Tabelle ist zu den Gitterweite $h = 2^{-3},\ 2^{-4}$ die Anzahl der Iterationen eingetragen, die nötig sind, um die angegebenen Fehlertoleranzen $\tau = 10^{-3},\ 10^{-4}$ einzuhalten.

	$\tau = 10^{-3}$	$\tau = 10^{-4}$	$\tau = 10^{-5}$
$h = 2^{-3}$	154	207	
$h = 2^{-4}$	620	830	
$h = 2^{-5}$			

Ergänzen Sie diese Tabelle und schätzen Sie insbesondere die Anzahl der Iterationen, die notwendig ist, um bei Schrittweite $h = 2^{-5}$ die Fehlertolerenz $\tau = 10^{-5}$ einzuhalten.

Hinweis: Die Anzahl der Unbekannten/Gleichungen ist proportional zu $\frac{1}{h^2}$

d) Schätzen Sie nun für dieses Problem die Anzahl der Iterationen, die nötig sind wenn man das Sytem mit dem GAUSS-SEIDEL-Verfahren iterativ löst und zwar für Schrittweiten $h = 2^{-3},\ 2^{-4},\ 2^{-5}$ und Fehlertoleranzen $\tau = 10^{-3},\ 10^{-4},\ 10^{-5}$.

	$\tau = 10^{-3}$	$\tau = 10^{-4}$	$\tau = 10^{-5}$
$h = 2^{-3}$			
$h = 2^{-4}$			
$h = 2^{-5}$			

e) Nun wird das Problem mit dem **SOR**-Verfahren iterativ gelöst und zwar jeweils unter Verwendung des optimalen Relaxationsparameter ω_{opt}.

Ergänzen in der folgenden Tabelle die fehlenden fünf Wert (wiederum jeweils die geschätzte Anzahl der Iterationen, die nötig ist um bei Gitterweite h die Fehlertoleranz τ einzuhalten).

	$\tau = 10^{-3}$	$\tau = 10^{-4}$	$\tau = 10^{-5}$
$h = 2^{-3}$	13	18	
$h = 2^{-4}$	25	35	
$h = 2^{-5}$			

6 Konjugierte Richtungen, nichtlineare Optimierung (9 Punkte)

a) Betrachten Sie die Matrix $\mathbf{A} = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix}$. Bestimmen Sie 3 Vektoren $\vec{r}_1$, $\vec{r}_2$, $\vec{r}_3$ die paarweise zueinander **A**-konjugiert sind.

Im Weiteren soll folgende Kostenfunktion (Zielfunktional) minimiert werden.

$$F(x, y) = x^2 + \frac{y^4}{2} - 2\,xy + 3\,y - 4\,x + 2$$

b) Führen Sie einen Schritt des **Gradienten-Verfahrens** durch. Wählen Sie als Startwert $[x_0, y_0] = [3, 1]$ und als Schrittweite $t = \frac{1}{4}$.

Erinnerung: Die folgende Kostenfunktion (Zielfunktional) soll minimiert werden.

$$F(x, y) = x^2 + \frac{y^4}{2} - 2\,xy + 3\,y - 4\,x + 2$$

c) Führen Sie nun einen Schritt des NEWTON-**Verfahrens** durch. Wählen Sie als Startwert $[x_0, y_0] = [3, 1]$.

Hinweis: Die Inverse einer 2-Matrix $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$ *ist gegeben durch* $\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad-bc} \cdot \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$

d) Nennen Sie zwei Abbruchkriterien für Iterationsverfahren.

7 Least Squares Approximation (7 Punkte)

a) Gegeben sind Datenpunkte x_i und zugehörige Datenwerte y_i

x_i	-1	0	2	3
y_i	0	0	2	2

Bestimmen Sie die Gerade $y = m \cdot x + t$ die diese Daten im *least squares* Sinne am besten approximiert.

b) Das überbestimmte lineare Gleichungssystem $\mathbf{A}\vec{x} = \vec{b}$ mit

$$\mathbf{A} = \begin{bmatrix} 5 & 4 & 5 \\ 3 & 4 & -2 \\ 1 & -2 & 3 \\ 1 & 6 & -4 \end{bmatrix}, \qquad \vec{b} = \begin{bmatrix} 1 \\ 7 \\ 7 \\ 3 \end{bmatrix} \quad \text{und unbekanntem} \quad \vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

soll im least squares Sinne gelöst werden.

Dabei ist die QR-Zerlegung von $\mathbf{A}$ bekannt, es gilt:

$$\mathbf{A} = \mathbf{QR} \qquad \text{wobei} \qquad \mathbf{Q} = \frac{1}{6} \cdot \begin{bmatrix} 5 & -1 & 3 & -1 \\ 3 & 1 & -5 & -1 \\ 1 & -3 & -1 & 5 \\ 1 & 5 & 1 & 3 \end{bmatrix} \quad \text{und} \quad \mathbf{R} = \begin{bmatrix} 6 & 6 & 3 \\ 0 & 6 & -6 \\ 0 & 0 & 3 \\ 0 & 0 & 0 \end{bmatrix}$$

Lösen Sie das überbestimmte Gleichungssystem $\mathbf{A}\vec{x} = \vec{b}$ unter Verwendung der QR-Zerlegung.

8 Programmierung: Polynominterpolation (11 Punkte)

In dieser Aufgabe soll eine unbekannte Funktion $f(x)\colon \mathbb{R} \to \mathbb{R}$ mittels einiger Abtastpunkte $(x_i, y_i)^T$, i = 1, ..., n durch ein Polynom $p(x)$ des Grades $n-1$ interpoliert werden. Die Implementierung erfolgt in `C++`. Im Gegensatz zu den Übungsaufgaben ist **keine** Fehlerbehandlung erforderlich.

Zur Interpolation der Samples soll die Monombasis für Polynome verwendet werden, d.h.

$$f(x) \approx p(x) = \sum_{i=0}^{n-1} a_i x^i.$$

Zum Bestimmen der Monomkoeffizienten muss bekanntermaßen ein Gleichungssystem $\mathbf{A}\vec{x} = \vec{b}$ gelöst werden, wobei der Lösungsvektor $\vec{x}$ die Koeffizienten enthält.

Hinweis: Sie finden die Klassenstrukturen auch auf der Seite 29, die zum Heraustrennen vorgesehen ist.

Die Klasse `Point2d`, die einen 2D-Punkt $(x, y)^T$ darstellt, ist vorgegeben und hat folgende Struktur:

```
class Point2d {
public:
  float x, y;                    //! Interne Darstellung
  ...                            //! Konstruktoren und Operatoren
};
```

Eine $m \times n$-Matrix wird von der Klasse `Matrix` repräsentiert:

```
class Matrix {
public:
  Matrix(unsigned int height, unsigned int width);      //! Konstruktor

  //! Operatoren zum Lese- und Schreibzugriff (m: Zeile, n: Spalte)
  float& operator()(unsigned int m, unsigned int n);
  float  operator()(unsigned int m, unsigned int n) const;

  unsigned int getHeight() const;        //! Anzahl der Zeilen
  unsigned int getWidth() const;         //! Anzahl der Spalten
  ...                                    //! weitere Methoden und Operatoren
};
```

Die Klasse `MonomCurve` repräsentiert ein Polynom in Monomdarstellung:

```
class MonomCurve {
public:
  MonomCurve(const std::vector<float>& coefficients);    //! Konstruktor

  float EvaluateAt(float x) const;               //! Auswerten an Stelle x
  float Integral(float a, float b) const;        //! Wert des Integrals von a bis b

  //! statische Methoden zum Aufstellen des Gleichungssystems
  static Matrix ComputeSystemMatrix(const std::vector<Point2d>& controlPoints);
  static Matrix ComputeRHS(const std::vector<Point2d>& controlPoints);

private:
  std::vector<float> m_coefficients;   //! Interne Darstellung (Monomkoeffizienten)
};
```

Hinweis: Verändern Sie die Klassenstrukturen nicht, d.h. führen Sie keine neuen Attribute oder Methoden ein.

a) Implementieren Sie die Methode `Matrix ComputeSystemMatrix(const std::vector<Point2d>& controlPoints, unsigned int degree)`, die die Systemmatrix $\mathbf{A}$ zur Bestimmung der Polynomkoeffizienten aufstellt. Die Methode erhält als Eingabeparameter die Samplepunkte (`controlPoints`) und soll die Matrix $\mathbf{A}$ zurückgeben.

Hinweis: Sie können die Anzahl der Elemente in einem `std::vector` mit der Methode `size()` abrufen.

```
Matrix MonomCurve::ComputeSystemMatrix(const std::vector<Point2d>& controlPoints) {




}
```

b) Implementieren Sie nun die Methode `Matrix ComputeRHS(const std::vector<Point2d>& controlPoints)`, die die rechte Seite $\vec{b}$ des Gleichungssystems bestimmt. Die Methode erhält als Eingabeparameter die Samplepunkte (`controlPoints`) und soll den Vektor $\vec{b}$ zurückgeben.

Hinweis: Ein Vektor $\vec{b} \in \mathbb{R}^n$ kann als Matrix der Größe $n \times 1$ implementiert werden.

```
Matrix MonomCurve::ComputeRHS(const std::vector<Point2d>& controlPoints) {




}
```

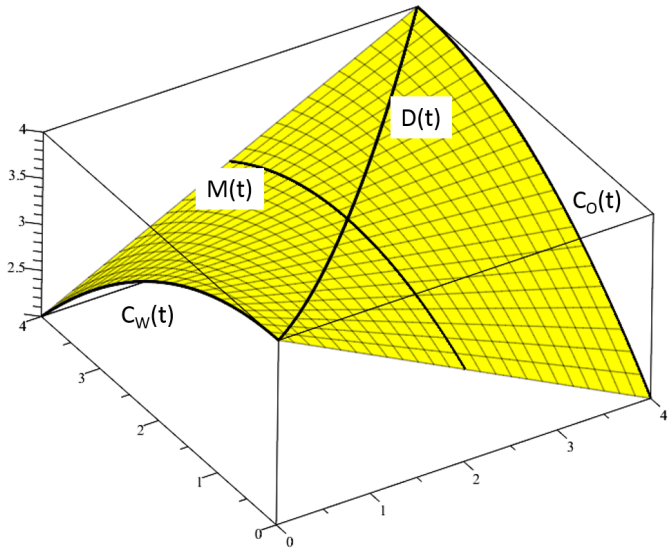
c) Implementieren Sie die Methode `float Integral(float a, float b)`, die das Integral $\int_a^b p(x)dx$ analytisch berechnet. Bilden Sie hierzu zunächst die Stammfunktion des Polynoms und werten Sie dieses neue Polynom an den entsprechenden Stellen aus.

Tipp: Sie können innerhalb der Methode ein neues Objekt des Typs `MonomCurve` anlegen und zur Auswertung verwenden.

```
float MonomCurve::Integral(float a, float b) const {
    float result;

    return result;
}
```

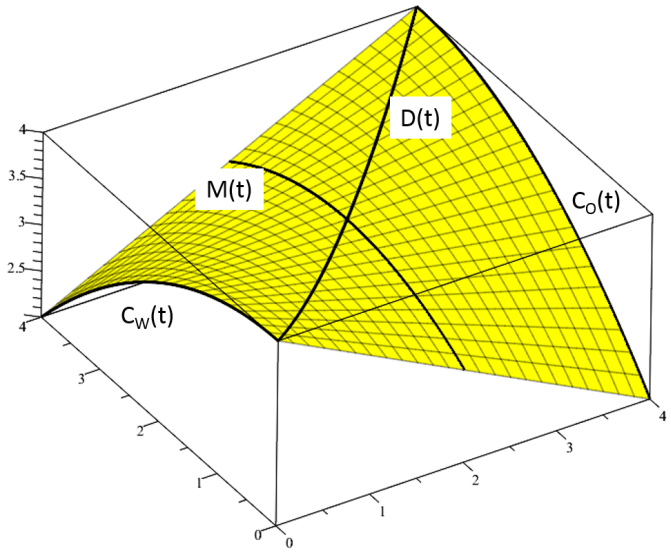
9 Bézier-Kurven (8 Punkte)



a) Betrachten Sie die quadratischen Bézier-Kurven

$$C_W(t) = \begin{bmatrix}0\\0\\4\end{bmatrix} \cdot B_0^2(t) + \begin{bmatrix}0\\2\\4\end{bmatrix} \cdot B_1^2(t) + \begin{bmatrix}0\\4\\2\end{bmatrix} \cdot B_2^2(t)\ , \quad C_O(t) = \begin{bmatrix}4\\0\\2\end{bmatrix} \cdot B_0^2(t) + \begin{bmatrix}4\\2\\4\end{bmatrix} \cdot B_1^2(t) + \begin{bmatrix}4\\4\\4\end{bmatrix} \cdot B_2^2(t)\ ,$$

Bestimmen Sie mit Hilfe des Algorithmus von deCasteljau die Kurvenpunkte $C_W\left(\frac{1}{4}\right)$ und $C_O\left(\frac{1}{4}\right)$.



b) Die beiden Randkurven werden linear kombiniert. Auf diese Weise entsteht die folgenden Fläche

$$S(s,t) = (1-s) \cdot C_W(t) + s \cdot C_O(t)$$

Die Mittelinie $M(t) = S\left(\frac{1}{2}, t\right)$ dieser Fläche ist eine quadratische Bézier-Kurve. Bestimmen Sie die Kontrollpunkte dieser Kurve.

c) Bestimmen Sie den Flächenpunkt $S\left(\frac{1}{2}, \frac{1}{4}\right)$.

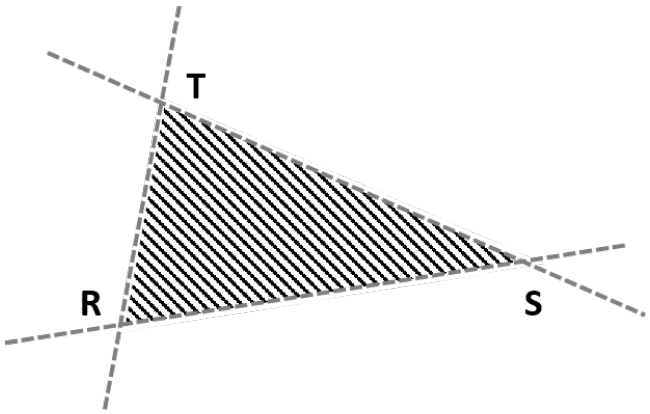
d) Die Diagonale $D(t) = S(t,t)$ dieser Fläche ist eine kubische Bézier-Kurve, $D(t) = \sum_{i=0}^{3} \vec{d}_i B_i^3(t)$.
Bestimmen Sie die beiden Kontrollpunkte $\vec{d_0}$ und $\vec{d_3}$ dieser Kurve.

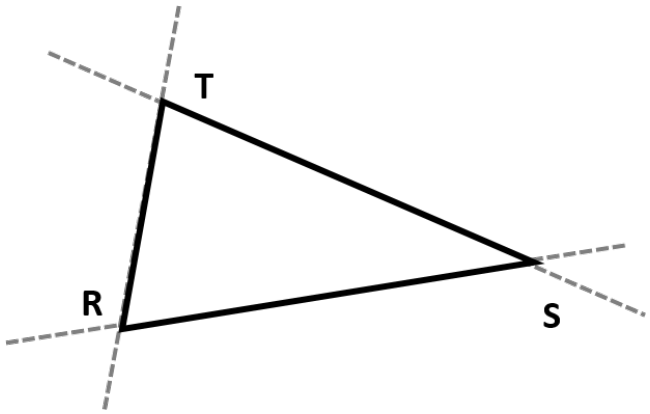
10 Multivariate Interpolation (9 Punkte)

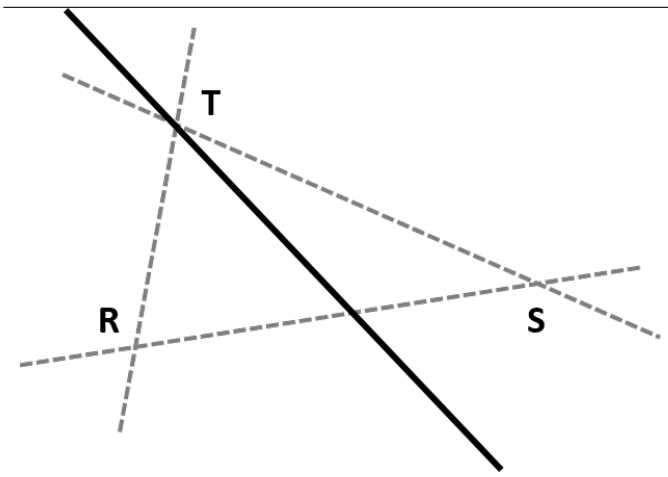
10.1 Baryzentrische Koordinaten

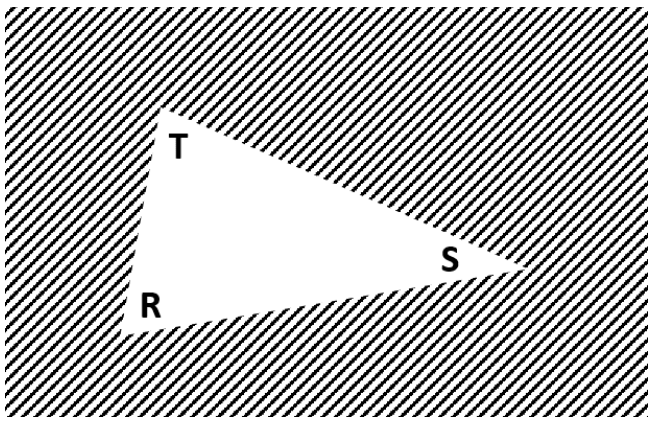
Gegeben sei jeweils ein Dreieck $\Delta(R, S, T)$ sowie die zugehörigen baryzentrischen Koordinaten $[\rho, \sigma, \tau]$. Dabei ist jeweils ρ das Gewicht von R, σ das Gewicht von S und τ das Gewicht von T.

a) Bestimmen Sie den Wertebereich der baryzentrischen Koordinaten $[\rho, \sigma, \tau]$ für folgende Gebiete (gestrichelte Linien gehören nicht zu einem Gebiet):

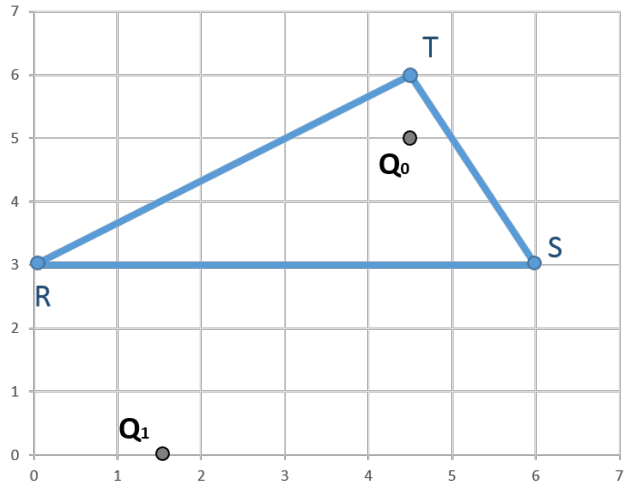






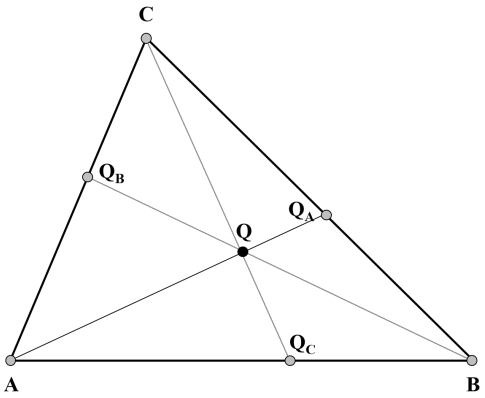


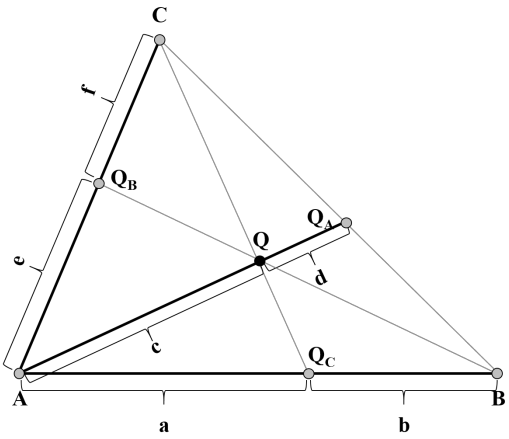
Im Nachfolgenden sei $R = [0, 3]^T$, $S = [6, 3]^T$ und $T = [4.5, 6]^T$.



b) Bestimmen Sie die baryzentrischen Koordinaten der Punkte $Q_0 = [4.5, 5]^T$ und $Q_1 = [1.5, 0]^T$ bezüglich des Dreiecks $\Delta(R, S, T)$.

c) Der Punkt Q liegt im Dreieck $\Delta(A, B, C)$ und seine baryzentrischen Koordinaten α, β und γ bzgl. $\Delta(A, B, C)$ seien bekannt (siehe linke Abb.).





Mit Hilfe der baryzentrischen Koordinaten von **Q** sollen die Teilungsverhältnisse der Strecken $\overline{AB}$, $\overline{AQ_A}$ und $\overline{AC}$ (siehe rechte Abb.) bestimmt werden.

Drücken Sie diese Teilungsverhältnisse mit Hilfe der baryzentrischen Koordinaten α, β, γ aus:

$$a : b \quad = \quad \boxed{:}$$

$$c : d \quad = \quad \boxed{:}$$

$$e : f \quad = \quad \boxed{:}$$

10.2 Bilineare Interpolation

Im Rechteck mit den Eckpunkten $P_{00} = [-1, -1]^T$, $P_{10} = [5, -1]^T$, $P_{11} = [5, 3]^T$ und $P_{01} = [-1, 3]^T$ sind die Werte $f_{00} = 0$, $f_{10} = 6$, $f_{11} = 2$ bzw. $f_{01} = 6$ gegeben. Diese Werte werden ins Innere bilinear interpoliert.

Bestimmen Sie die Werte des Interpolanten im Punkt $P = [1, 2]^T$ sowie im Mittelpunkt $M = \frac{1}{4}(P_{00} + P_{10} + P_{11} + P_{01})$ des Rechtecks.

Hinweis: Eine Skizze ist unter Umständen hilfreich.

Anhang zur Programmierung: Klassenübersicht

Hinweis: Sie dürfen dieses Blatt gerne aus der Klausur heraustrennen. Es wird **nicht** bewertet.

```
class MedianCut {
public:
 //! Klassen Konstruktor
 MedianCut(const std::vector<Point2d>& pointcloud);

 //! Einstiegspunkt zur Berechnung des Median Cut
 std::vector<Point2d> ComputeMedianCut(int nCuts);

private:
 //! Berechne Bounding Box (min, max) der übergebenen Punktewolke pc
 static void ComputeBoundingBox(const std::vector<Point2d>& pc,
                                Point2d& min, Point2d& max);

 //! Sortiert die übergebene Punktewolke entlang der X- bzw. Y-Achse
 static void SortPointCloudAlongXAxis(std::vector<Point2d>& pointcloud);
 static void SortPointCloudAlongYAxis(std::vector<Point2d>& pointcloud);

 //! Gibt den Repräsentanten der Punkte zurück (hier: Mittelwert)
 static Point2d GetRepresentative(const std::vector<Point2d>& pointcloud);

 //! Rekursive Berechnung des Median Cut
 //! (nCuts: maximale Anzahl der Schritte, cCuts: aktueller Schritt)
 static void MedianCutRecursion(std::vector<Point2d> subpointcloud,
                        int cCuts, int nCuts, std::vector<Point2d>& result);

 //! Klassenvariable
 std::vector<Point2d> m_pointcloud;
};
```

```
class Point2d {
public:
  float x, y;                    //! Interne Darstellung
  Point2d(float x, float y);     //! Konstruktor
  ...                            //! weitere Konstruktoren und Operatoren
};
```

Hinweis: Sie können davon ausgehen, dass für Objekte der Klasse `Point2d` alle arithmetischen Standardoperatoren (`+, -, *, /, =, ==, !=, +=, -=, *=, /=`) zur Verfügung stehen.

Hinweis: Sie dürfen dieses Blatt gerne aus der Klausur heraustrennen. Es wird **nicht** bewertet.

```
class Matrix {
public:
  //! Konstruktor
  Matrix(unsigned int height, unsigned int width);

  //! Operatoren zum Lese- und Schreibzugriff (m: Zeile, n: Spalte)
  float& operator()(unsigned int m, unsigned int n);
  float  operator()(unsigned int m, unsigned int n) const;

  //! Anzahl der Zeilen
  unsigned int getHeight() const;

  //! Anzahl der Spalten
  unsigned int getWidth() const;

  ...                              //! weitere Methoden und Operatoren
};
```

```
class MonomCurve {
public:
  //! Konstruktor
  MonomCurve(const std::vector<float>& coefficients);

  //! Auswerten an Stelle x
  float EvaluateAt(float x) const;

  //! Wert des Integrals von a bis b
  float Integral(float a, float b) const;

  //! statische Methode zum Aufstellen der Systemmatrix A
  static Matrix ComputeSystemMatrix(const std::vector<Point2d>& controlPoints);

  //! statische Methode zum Aufstellen der rechten Seite b
  static Matrix ComputeRHS(const std::vector<Point2d>& controlPoints);

private:
  //! Interne Darstellung (Monomkoeffizienten)
  std::vector<float> m_coefficients;
};
```