TN
Teilnehmernummer
(von Platzkarte)

Lehrstuhl für Graphische Datenverarbeitung • Friedrich-Alexander Universität Erlangen-Nürnberg
Prof. Dr. Günther Greiner

14. Februar 2017

Klausur
Algorithmik kontinuierlicher Systeme – 14. Februar 2017

Angaben zur Person (Bitte in DRUCKSCHRIFT ausfüllen!):

Name ..... Vorname .....

Mat.-Nummer .....

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

- Die Bearbeitungszeit beträgt **90 Minuten**.
- Hilfsmittel (außer Schreibmaterial und Taschenrechner) sind nicht zugelassen.
- **Überprüfen Sie die Prüfungsangaben auf Vollständigkeit (27 Seiten inklusive Deckblatt) und einwandfreies Druckbild!**
- Schreiben Sie deutlich und ausschließlich mit blauer oder schwarzer Tinte. Unleserliche Antworten gehen **nicht** in die Bewertung ein. Benutzen Sie **nur für Zeichnungen** einen Bleistift.
- Sollte der Platz nicht reichen, so verwenden Sie zunächst die Zusatz-Seiten am Ende der Klausur. Fügen Sie einen Hinweis in Ihre Lösung ein, dass die Lösung auf den Zusatz-Seiten fortgesetzt wurde. Sollte der Platz immer noch nicht ausreichen, so müssen Sie bei der Aufsicht weitere Zusatz-Seiten anfordern und einheften lassen. **Es werden nur leere Blätter eingeheftet!**
- Falls Sie zusätzliches Schmierpapier benötigen, melden Sie sich bitte bei der Aufsicht.
  Das Schmierpapier wird eingesammelt, aber nicht bewertet!
- Wenn Sie die Prüfung aus gesundheitlichen Gründen abbrechen müssen, so muss Ihre Prüfungsunfähigkeit durch eine ärztliche Untersuchung nachgewiesen werden. Melden Sie sich bei der Aufsicht und lassen Sie sich das entsprechende Formular aushändigen.
- Die angegebene Punkteverteilung gilt unter Vorbehalt.

Erklärung

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

Erlangen, 14. Februar 2017 .....
(Unterschrift)

Nicht von der Kandidatin bzw. vom Kandidaten auszufüllen !

Aufgabe	1	2	3	4	5	6	7	8	9	10	**Gesamt**
Max. Punktzahl	6	5	10	7	10	10	6	16	9	11	90
Erreichte Punkte											
Zweitkorrektor											

1 Theoriefragen (6 Punkte)

a) Beantworten Sie die folgenden Fragen! Schreiben Sie ihre Antwort in die rechte Spalte der Tabelle! Gehen Sie davon aus, dass $\mathbf{A}$ eine $(n \times n)$-Matrix ist und $\vec{x},\ \vec{y} \in \mathbb{R}^n$.

Welche Komplexität hat die Berechnung des Skalarproduktes $\vec{y}^T\vec{x}$?	$\mathcal{O}($ $)$
Welche Komplexität hat die Matrix-Vektor-Multiplikation $\mathbf{A}\vec{x}$ wenn $\mathbf{A}$ eine k-diagonale Matrix ist?	$\mathcal{O}($ $)$
Welche Komplexität hat die Bestimmung von $\mathbf{A}^{-1}$ einer Diagonalmatrix mit vollem Rang?	$\mathcal{O}($ $)$
Wie viele Jacobi-Rotation benötigt die Bestimmung der **QR**-Zerlegung für eine tridiagonale Matrix?	
Wie groß ist der Approximationsfehler des Catmull-Rom-Interpolanten bei Schrittweite h?	$\mathcal{O}($ $)$
Wie groß ist der Approximationsfehler der iterierten Simpson-Regel mit Schrittweite h?	$\mathcal{O}($ $)$
Wieviele Kontrollpunkte entstehen bei einem midpoint-subdivision Schritt einer Bézier-Kurve vom Grad n ?	
Welche Komplexität hat die Bestimmung eines einzelnen Punktes auf einer Bézier-Kurve mit n Kontrollpunkten mit dem Algorithmus von De Casteljau?	$\mathcal{O}($ $)$

b) Sind folgende Gleichungssysteme $\mathbf{A}\vec{x} = \vec{b}$ überbestimmt, unterbestimmt oder eindeutig lösbar?

$\begin{bmatrix}1&0&0\\0&2&0\\0&0&3\end{bmatrix}\vec{x} = \begin{bmatrix}3\\2\\1\end{bmatrix}$	$\begin{bmatrix}1&2\\0&2\\0&3\end{bmatrix}\vec{x} = \begin{bmatrix}3\\2\\1\end{bmatrix}$	$\begin{bmatrix}1&2&0\\0&2&3\\2&4&0\end{bmatrix}\vec{x} = \begin{bmatrix}2\\1\\4\end{bmatrix}$	$\begin{bmatrix}1&2\\0&2\\2&4\end{bmatrix}\vec{x} = \begin{bmatrix}1\\4\\2\end{bmatrix}$

2 Dünnbesetzte Matrizen (5 Punkte)

a) Speichern Sie die folgende Matrix **A** im **CRS − Format** (Compressed Row Storage) ab.
Hinweis: Die Indizierung beginnt bei 1.

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 7 & 0 \\ -2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

b) Gegeben ist nun ein Vektor $\vec{b} = [1, 1, -1, 1, 0]^T$ und eine Matrix **B** im **CCS − Format** (Compressed Column Storage):

$$\begin{aligned} \texttt{val} &= [1, 4, 7, -2, 5] \\ \texttt{row\_ind} &= [2, 3, 4, 2, 3] \\ \texttt{col\_ptr} &= [1, 1, 2, 4, 6] \end{aligned}$$

Bestimmen Sie den Vektor $\vec{b}^T\mathbf{B}$ ohne die Matrix zu rekonstruieren.
Hinweis: Zwischenschritte müssen erkennbar sein! Die Indizierung beginnt bei 1.

c) Wie kann man eine $(n \times n)$-Tridiagonalmatrix möglichst effizient speichern?

3 Programmierung: LR-Zerlegung (10 Punkte)

Die Klasse `Solver` stellt grundlegende Funktionen zum Lösen von linearen Gleichungssystemen bereit. Dazu wird die Klasse `Matrix` verwendet. Sie sollen dabei einige Methoden der Klasse `Solver` in `C++` implementieren. Entgegen der Übungen ist **keine** Fehlerbehandlung erforderlich. Verändern Sie die Klassenstrukturen nicht, d.h. führen Sie keine neuen Attribute / Methoden ein.

```
class Solver {
public:
    //! Berechnet eine LR-Zerlegung für die quadratische Matrix A
    static void decomposeA(const Matrix &A, Matrix &L, Matrix &R);

    //! Berechnet Ly = b, wobei y das Ergebnis ist
    //! L ist eine untere Dreiecksmatrix
    static void forwardSubstitution(const Matrix &L, const Matrix &b, Matrix &y);

    //! Berechnet Rx = y, wobei x das Ergebnis ist
    //! R ist eine obere Dreiecksmatrix
    static void backwardSubstitution(const Matrix &R, const Matrix &y, Matrix &x);

    //! Loest das lineare System Ax = b mittels LR-Zerlegung
    static void solveSystem(const Matrix &A, const Matrix &b, Matrix &x);

    //! Berechnet die Determinante der Matrix A
    static float calcDeterminant(const Matrix &A);
};
```

Die Methode `decomposeA(const Matrix &A, Matrix &L, Matrix &R)` berechnet für die Matrix **A** eine LR-Zerlegung, die die folgende allgemeine Struktur besitzt:

$$\underbrace{\begin{bmatrix} \star & \cdots & \cdots & \star \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ \star & \cdots & \cdots & \star \end{bmatrix}}_{\mathbf{A}} = \underbrace{\begin{bmatrix} 1 & 0 & \cdots & 0 \\ \star & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ \star & \cdots & \star & 1 \end{bmatrix}}_{\mathbf{L}} \cdot \underbrace{\begin{bmatrix} \star & \cdots & \cdots & \star \\ 0 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & \star \end{bmatrix}}_{\mathbf{R}}$$

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

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

    //! Mutator (i = Zeile, j = Spalte)
    float& operator()(unsigned int i, unsigned int j);

    //! Akzessor (i = Zeile, j = Spalte)
    float operator()(unsigned int i, unsigned int j) const;

    ...  // weitere Konstruktoren und Methoden
};
```

a) Implementieren Sie die Methode `void backwardSubstitution(const Matrix &R, const Matrix &y, Matrix &x)`, die das Gleichungssystem $\mathbf{R}\vec{x} = \vec{y}$ löst.

Die Methode erhält als Eingabeparameter eine obere Dreiecksmatrix $\mathbf{R}$ und einen Vektor $\vec{y}$, der als $n \times 1$ Matrix dargestellt wird. Die Lösung wird in den Vektor $\vec{x}$ geschrieben, der ebenfalls als $n \times 1$ Matrix dargestellt wird. Sie können davon ausgehen, dass die Ergebnismatrix $\vec{x}$ bereits die richtige Größe hat.

```
void Solver::backwardSubstitution(const Matrix &R, const Matrix &y, Matrix &x) {



}
```

b) Implementieren Sie die Methode `void solveSystem(const Matrix &A, const Matrix &b, Matrix &x)`, die das Gleichungssystem $\mathbf{A}\vec{x} = \vec{b}$ mittels LR-Zerlegung löst.

Die Methode erhält als Eingabeparameter eine quadratische Matrix $\mathbf{A}$ und einen Vektor $\vec{b}$, der als $n \times 1$ Matrix dargestellt wird. Die Lösung wird in den Vektor $\vec{x}$ geschrieben, der ebenfalls als $n \times 1$ Matrix dargestellt wird. Verwenden Sie dazu passende Methoden der Klasse `Solver`.

Sie können davon ausgehen, dass die Matrix $\mathbf{A}$ quadratisch ist und $\vec{x}$ bereits die richtige Größe hat.

```
void Solver::solveSystem(const Matrix &A, const Matrix &b, Matrix &x) {



}
```

c) Implementieren Sie die Methode `float calcDeterminant(const Matrix &A)`, die die Determinante der Matrix **A** berechnet. Nutzen Sie eine LR-Zerlegung aus und verwenden Sie dazu passende Methoden der Klasse `Solver`. Sie können davon ausgehen, dass die Matrix **A** quadratisch ist.

```
float Solver::calcDeterminant(const Matrix &A) {
    float result = 1.0f;









    return result;
}
```

d) Beschreiben Sie kurz, wie man die oben genannte Zerlegung möglichst effizient speichern kann.

4 QR-Zerlegung (7 Punkte)

a) Nennen Sie zwei Verfahren, wie man eine QR-Zerlegung bestimmen kann.

b) Bestimmen Sie die QR-Zerlegung der folgenden Matrix:

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

c) Die QR-Zerlegung der Matrix $\mathbf{B} = \begin{bmatrix} 1 & -3 & -4 & -2 \\ 1 & 1 & 0 & 2 \\ -1 & -1 & -4 & 0 \\ -1 & 3 & 0 & -4 \end{bmatrix}$ ist bekannt:

$$\mathbf{B} = \underbrace{\frac{1}{2}\begin{bmatrix} 1 & -1 & -1 & -1 \\ 1 & 1 & -1 & 1 \\ -1 & -1 & -1 & 1 \\ -1 & 1 & -1 & -1 \end{bmatrix}}_{Q} \underbrace{\begin{bmatrix} 2 & -2 & 0 & 2 \\ 0 & 4 & 4 & 0 \\ 0 & 0 & 4 & 2 \\ 0 & 0 & 0 & 4 \end{bmatrix}}_{R}$$

Bestimmen Sie unter Verwendung der QR-Zerlegung von $\mathbf{B}$ die Lösung der linearen Gleichung

$$\mathbf{B}\vec{x} = \vec{b} \quad \text{für} \quad \vec{b} = [\,-6\,,\ 6\,,\ -2\,, -6\,]^T \ .$$

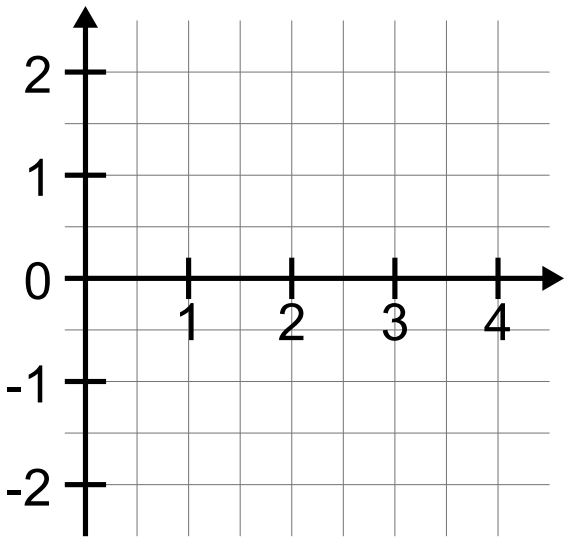
d) Bestimmen Sie für die Matrix $\mathbf{B}$ aus Teil c) den Betrag der Determinante.

5 Interpolation (10 Punkte)

Gegeben seien folgende Punkte:

i	0	1	2	3
x_i	0	1	2	4
y_i	1	-1	1	2

a) **Zeichnen** Sie die Funktion $n(x) : [0, 4] \mapsto \mathbb{R}$, welche obige Werte, gemäß Nearest Neighbor Interpolation, stückweise konstant interpoliert.



Nearest neighbor interpolant

b) **Berechnen** Sie die Funktion $l(x) : [0, 4] \mapsto \mathbb{R}$, welche obige Werte stückweise linear interpoliert.

Zur Erinnerung:

i	0	1	2	3
x_i	0	1	2	4
y_i	1	-1	1	2

c) Bestimmen Sie die LAGRANGE-Polynome zu den Stützstellen und geben Sie die Koeffizienten des Interpolationspolynoms an.

Hinweis: Die Basis-Funktionen müssen nicht ausmultipliziert werden!

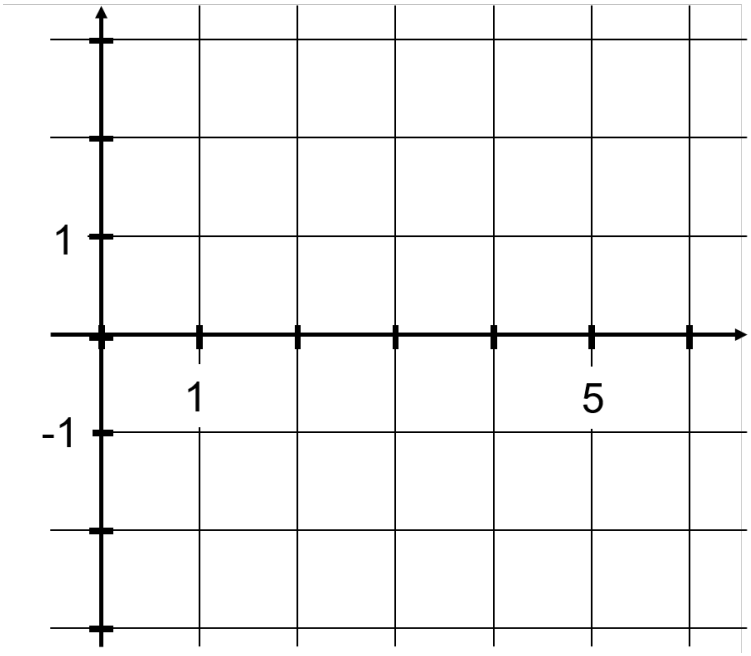
d) Bestimmen Sie die NEWTON-Polynome und geben Sie die Koeffizienten des Interpolationspolynoms an.
Hinweis: Die Basis-Funktionen müssen nicht ausmultipliziert werden!

In dieser und der nachfolgenden Teilaufgabe sind andere Interpolationsdaten gegeben und zwar

i	0	1	2	3
x_i	0	2	4	6
y_i	-1	-3	-1	3

e) Geben Sie die Ableitungen m_1 und m_2 des Catmull-Rom-Interpolanten an den Stellen x_1 und x_2 an.

f) Skizzieren Sie im Interval $[x_1, x_2]$ den Catmull-Rom-Interpolanten.



6 Iterative Lösungsverfahren (10 Punkte)

Gegeben seien die 3×3-Matrix $\mathbf{A}$ sowie der Vektor $\vec{b}$ mit

$$\mathbf{A} = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 1 \\ -1 & 1 & 2 \end{bmatrix} \qquad \vec{b} = \begin{bmatrix} 8 \\ 0 \\ 4 \end{bmatrix} .$$

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

b) Wie verhält sich (in den meisten praktischen Fällen) die Anzahl der Iterationsschritte beim Gauss-Seidel- zu der des Jacobi-Verfahrens bei gleicher Fehlertoleranz?

c) Mit welchem Verfahren kann man (bei positiv definiter Matrix) die Konvergenzgeschwindigkeit vom Gauss-Seidel-Verfahren steigern?

d) Implementieren Sie eine Methode `Matrix jacobi(const Matrix &A, const Matrix &b, unsigned int nIterations)`, die für eine übergebene **quadratische** Matrix **A** und einen Vektor $\vec{b}$ das JACOBI-Verfahren mit `nIterations` Iterationen ausführt. Der Vektor $\vec{b}$ wird als $(n \times 1)$-Matrix gespeichert.

Verwenden Sie `C++`-Syntax.

Hinweis: Die Indizierung beginnt bei 0. Wie in den Programmierübungen können auf die Matrix-Elemente mit dem `operator()(unsigned int row, unsigned int column)` zugegriffen werden, z.B. liefert `A(0, 1)` das Element $a_{0,1}$.

Entgegen der Übungen ist **keine Fehlerbehandlung** erforderlich.

```
Matrix jacobi(const Matrix &A, const Matrix &b, unsigned int nIterations) {
    unsigned int n = A.getHeight();        // A ist quadratisch (n x n)

    Matrix x(n, 1);









    return x;
}
```

e) Implementieren Sie eine Methode `bool isDiagonallyDominant(const Matrix &A)`, die überprüft, ob die **quadratische** Matrix **A** die Eigenschaft der **strikten** (starken) Diagonaldominanz erfüllt.

Verwenden Sie `C++`-Syntax.

Hinweis: Die Indizierung beginnt bei 0. Wie in den Programmierübungen können auf die Matrix-Elemente mit dem `operator()(unsigned int row, unsigned int column)` zugegriffen werden, z.B. liefert `A(0, 1)` das Element $a_{0,1}$.

Um den Absolutbetrag $|x|$ eines Skalars x zu berechnen, können Sie `abs(x)` verwenden.

Entgegen der Übungen ist **keine Fehlerbehandlung** erforderlich.

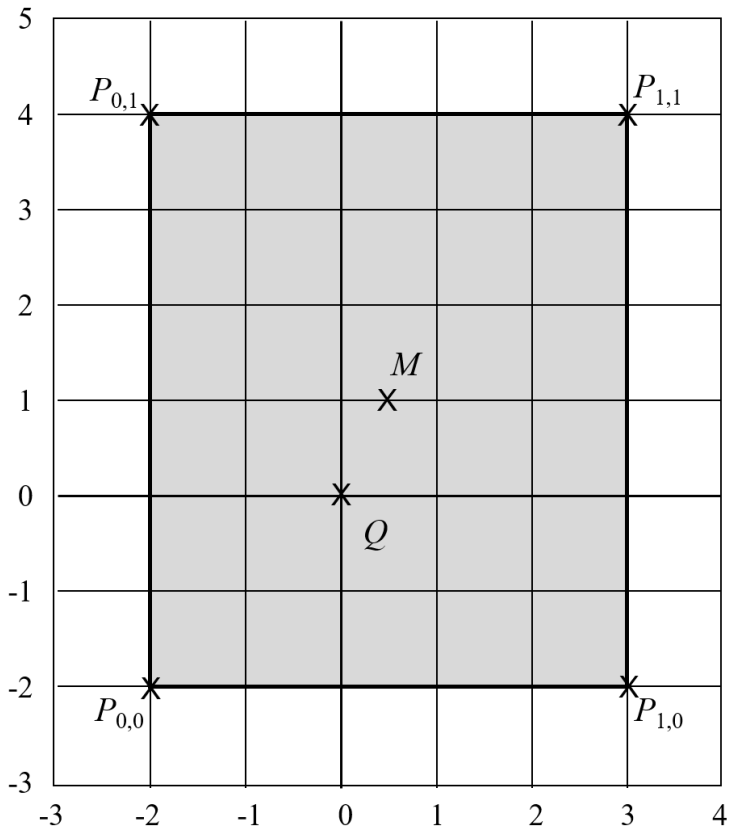
```
bool isDiagonallyDominant(const Matrix &A) {
    unsigned int n = A.getHeight();        // A ist quadratisch (n x n)
```

```
}
```

7 Multivariate Interpolation (6 Punkte)

7.1 Bilineare Interpolation

a) In den Ecken eines Rechtecks $P_{0,0} = [-2, -2],\ P_{1,0} = [3, -2],\ P_{1,1} = [3, 4],\ P_{0,1} = [-2, 4]$ sind vier Werte f_{00}, f_{10}, f_{11} und f_{01} gegeben, diese sollen bilinear interpoliert werden:



Die Werte des bilinearen Interpolanten in einem Punkt P kann man als gewichtete Summe schreiben: $f_P = w_{00}^P f_{00} + w_{10}^P f_{10} + w_{11}^P f_{11} + w_{01}^P f_{01}$

Bestimmen Sie die Gewichte für den Mittelpunkt des Rechtecks $M = [0.5, 1]$ und den Punkt $Q = [0, 0]$!

M: $w_{00}^M =$, $w_{10}^M =$, $w_{11}^M =$, $w_{01}^M =$;

Q: $w_{00}^Q =$, $w_{10}^Q =$, $w_{11}^Q =$, $w_{01}^Q =$;

7.2 Baryzentrische Koordinaten

Gegeben sind die fünf Punkte $A = \begin{bmatrix}-2\\-1\end{bmatrix}$, $B = \begin{bmatrix}6\\-1\end{bmatrix}$, $C = \begin{bmatrix}6\\3\end{bmatrix}$, $D = \begin{bmatrix}-2\\3\end{bmatrix}$ und $Q = \begin{bmatrix}0\\0\end{bmatrix}$.

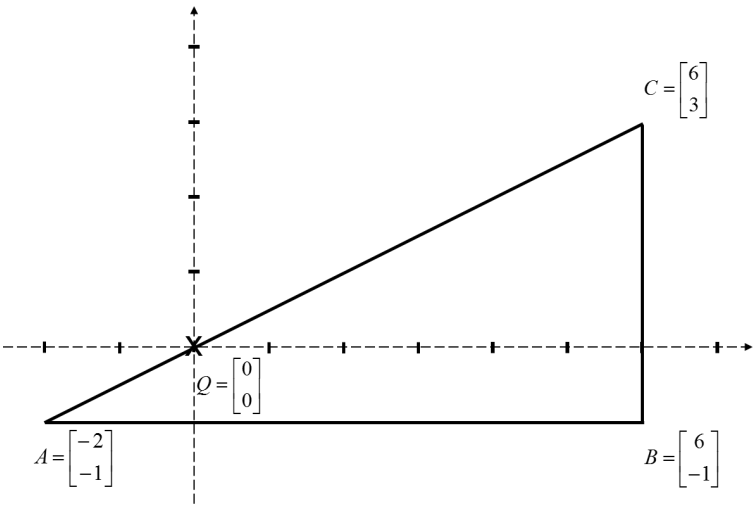
b) Bestimmen Sie die baryzentrischen Koordinaten des Punktes Q bezüglich der folgenden Dreiecke.
Tipp: Die Lösung kann geometrisch bestimmt werden.

- Bezüglich des Dreiecks $\Delta(A, B, C)$,
$Q = \alpha_1 A + \beta_1 B + \gamma_1 C$:

$\alpha_1 =$

$\beta_1 =$

$\gamma_1 =$

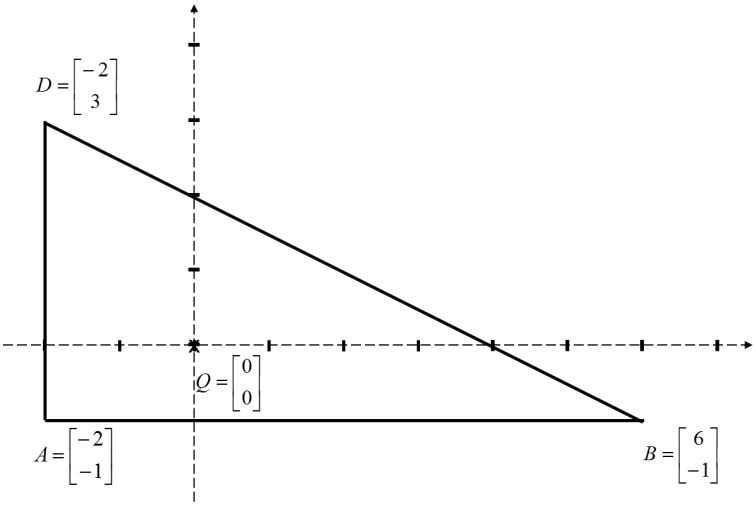


- Bezüglich des Dreiecks $\Delta(A, B, D)$
$Q = \alpha_2 A + \beta_2 B + \delta_2 D$:

$\alpha_2 =$

$\beta_2 =$

$\delta_2 =$

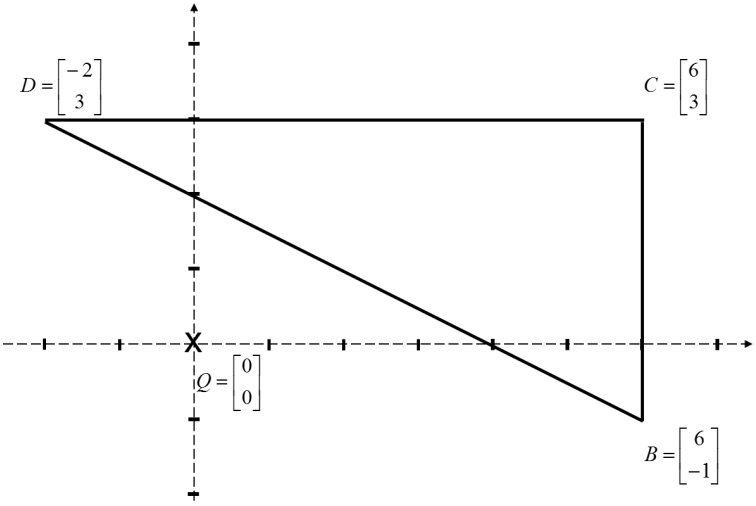


- Bezüglich des Dreiecks $\Delta(B, C, D)$
$Q = \beta_3 B + \gamma_3 C + \delta_3 D$:

$\beta_3 =$

$\gamma_3 =$

$\delta_3 =$



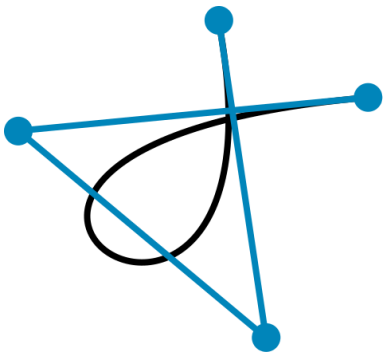
8 Bézier-Kurven (16 Punkte)

a) Betrachten Sie die Bézier-Kurve $C(t)\,,\ (0 \leq t \leq 1)$ mit den Kontrollpunkten

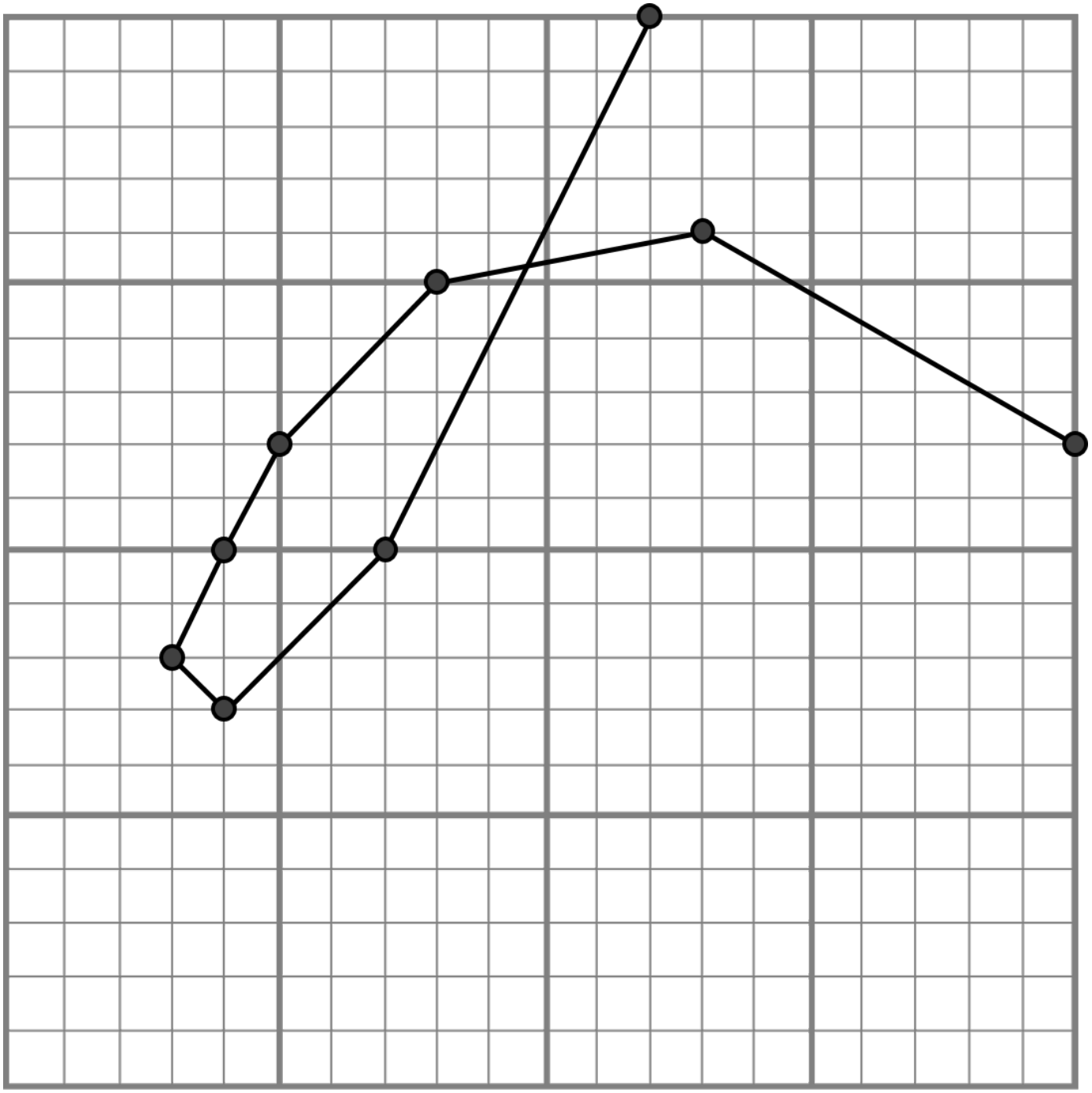
$$\vec{b}_0 = \begin{bmatrix} 56 \\ 32 \end{bmatrix}, \quad \vec{b}_1 = \begin{bmatrix} 0 \\ 8 \end{bmatrix}, \quad \vec{b}_2 = \begin{bmatrix} 8 \\ 0 \end{bmatrix}, \quad \vec{b}_3 = \begin{bmatrix} 16 \\ 8 \end{bmatrix}.$$

Werten Sie die Kurve $C(t)$ an der Stelle $t = \frac{3}{4}$ aus.

b) Nennen Sie **drei** Formeigenschaften von Bézier-Kurven und beurteilen Sie, ob diese in der nachfolgenden Abbildung erfüllt sind oder nicht.



c) Für eine BÉZIER-Kurve $B(t)$ wurde ein `midpoint subdivision` Schritt ausgeführt und die Kontrollpolygone der beiden Teilkurven in nachfolgender Abbildung gezeichnet. Bestimmen Sie die Kontrollpunkte $(\vec{c}_0, \vec{c}_1, \vec{c}_2, \vec{c}_3, \vec{c}_4)$ der Kurve $B(t)$ geometrisch und markieren Sie diese **deutlich** mit einem Kreuz.



d) In der folgenden Methode soll eine BÉZIER-Kurve $C(u)$, die durch die Kontrollpunkte `cp` beschrieben wird, mit einer affinen Abbildung $\phi(\vec{x})$ transformiert werden: $\tilde{C}(u) = \phi(C(u))$. Die affine Abbildung $\phi(\vec{x})$ wird definiert durch $\phi(\vec{x}) = \mathbf{A}\vec{x} + \vec{t}$. Geben Sie die Kontrollpunkte der transformierten Kurve $\tilde{C}(u)$ zurück.

Hinweis: Gehen Sie davon aus, dass für die Klassen `Point2D` und `Matrix2D` alle arithmetischen Operatoren überladen sind. Sie können alle Methoden der Standard-Template-Library (STL) verwenden, insbesondere `size()` und `push_back()`.

```
vector<Point2D> transform(const vector<Point2D> &cp, const Matrix2D &A, const Point2D &t) {




}
```

e) In den folgenden Methoden soll der **Midpoint-Subdivision-Algorithmus** in `C++` implementiert werden.

- Es gibt eine **rekursive Methode** `subdivide(...)` (siehe nächste Seite), die die Subdivision auf den aktuellen Kontrollpunkten `cp` durchführt, solange die aktuelle Rekursionstiefe `curDepth` die maximale Rekursionstiefe `maxDepth` nicht erreicht hat.
- Der **Einstieg in die Rekursion** soll in der Methode `evalSubdivision(...)` stattfinden.

Entgegen der Übungen ist keine Fehlerbehandlung erforderlich. Duplikate können ignoriert werden.

Hinweis: Gehen Sie davon aus, dass für die Klasse `Point2D` alle arithmetischen Operatoren überladen sind. Sie können alle Methoden der Standard-Template-Library (STL) verwenden, insbesondere `size()` und `push_back()`.

```
vector<Point2D> evalSubdivision(const vector<Point2D> &cp, unsigned int maxDepth) {




}
```

```
void subdivide(vector<Point2D> &res, int maxDepth, int curDepth, const vector<Point2D> &cp){
    unsigned int n = cp.size();        // n: Anzahl der Kontrollpunkte









}
```

9 SVD und Hauptkomponentenanalyse (9 Punkte)

a) Lösen Sie das lineare Gleichungssystem $\mathbf{A}\vec{x} = \vec{b}$ für $\vec{b} = [1, -9, 8]^T$ mit der gegebenen SVD von $\mathbf{A}$.

$$\mathbf{A} = \underbrace{\begin{bmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}}_{U} \underbrace{\begin{bmatrix} 9 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\Sigma} \underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}}_{V^T} .$$

b) Bestimmen Sie die Eigenwerte und Eigenvektoren von $\mathbf{A}^T\mathbf{A}$.
Hinweis: Verwenden Sie die Angaben aus Teilaufgabe a).

Gegeben seien die folgenden 2D-Datenpunkte $\vec{p}_i = \left[x_i, y_i\right]^T$:

$$\vec{p}_0 = \begin{bmatrix}4\\0\end{bmatrix},\ \vec{p}_1 = \begin{bmatrix}-1\\-1\end{bmatrix},\ \vec{p}_2 = \begin{bmatrix}1\\2\end{bmatrix},\ \vec{p}_3 = \begin{bmatrix}-1\\2\end{bmatrix},\ \vec{p}_4 = \begin{bmatrix}2\\2\end{bmatrix},\ \vec{p}_5 = \begin{bmatrix}1\\1\end{bmatrix}.$$

c) Bestimmen Sie die Kovarianzmatrix $\mathbf{C}$, die zu den $\vec{p}_i$ gehört.

d) Gegeben sei die folgende Kovarianzmatrix:

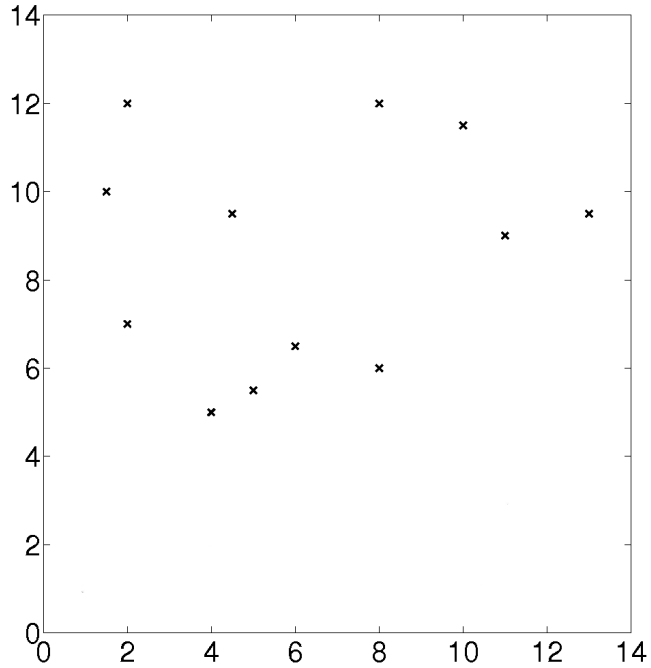
$$\mathbf{B} = \begin{bmatrix} 4 & -3 \\ -3 & 4 \end{bmatrix}.$$

Bestimmen Sie die zu $\mathbf{B}$ gehörenden Hauptachsen.

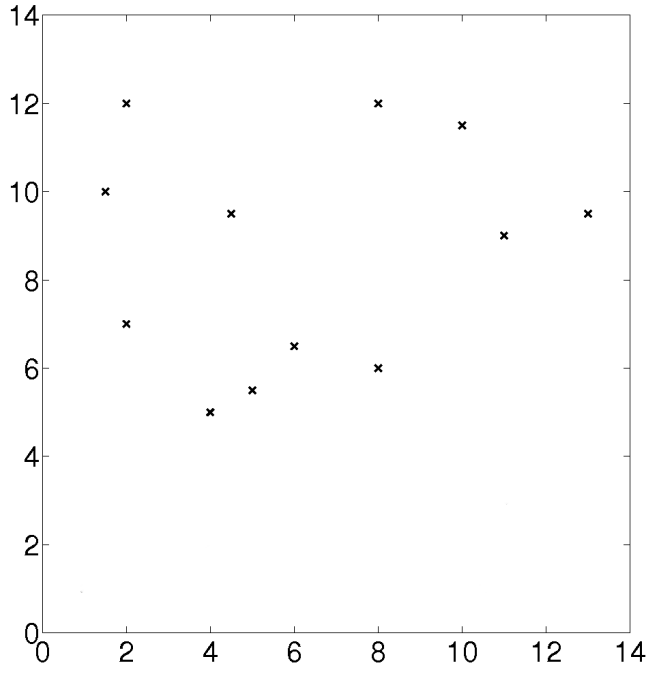
10 Gemischtes (11 Punkte)

10.1 Median Cut (4 Punkte)

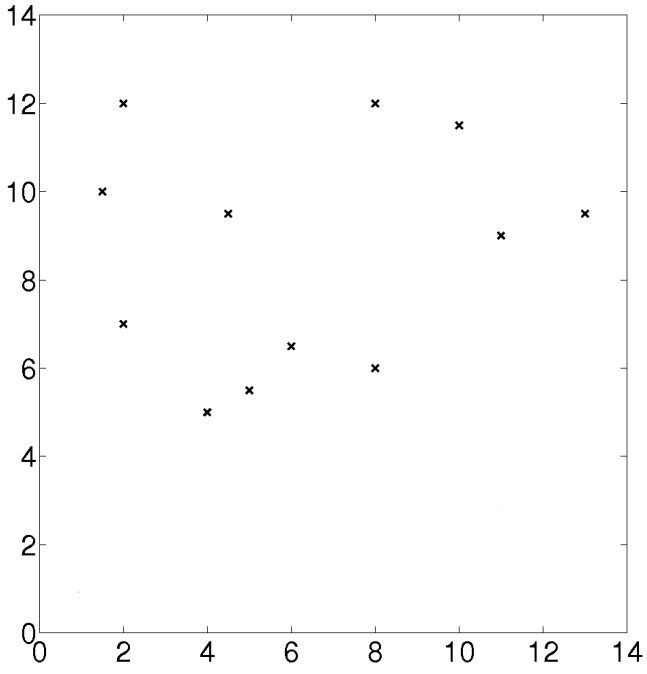
Gegeben ist eine Punktewolke mit 12 Punkten. Führen Sie 2 Schritte des **Median-Cut** Verfahrens durch. Benutzen Sie dazu die folgenden Vorlagen:



Gegebene Punktewolke



Schritt 1



Schritt 2

10.2 Kondition (3 Punkte)

Bestimmen Sie die Konditionszahl der folgenden Matrix $\mathbf{A} = \begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix}$, für $a = 10$, $a = 1$ und $a = \frac{1}{10}$ für eine von Ihnen gewählte Matrixnorm.
Geben Sie auch an, welche Matrixnorm Sie verwenden.

10.3 Nichtlineare Optimierung (4 Punkte)

Gegeben ist der Gradient $\nabla F(x, y) = [xy^2 + x + y - 4, x^2y + x + 2y - 8]^T$ der Funktion $F(x, y)$. Diese soll mit einem Abstiegsverfahren minimiert werden.

a) Führen Sie einen Schritt des Gradienten-Verfahrens mit Schrittweite $\tau = \frac{1}{2}$ und Startwert $[x_0, y_0]^T = [0, 0]^T$ durch.

b) Führen Sie einen Schritt des NEWTON-Verfahrens mit Startwert $[x_0, y_0]^T = [0, 0]^T$ durch.