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

12. Februar 2013

Algorithmik kontinuierlicher Systeme — 12. Februar 2013

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	4	9	11	7	8	7	9	10	10	15
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 in der Universitätsklinik 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 (28 Seiten inklusive Deckblatt) und einwandfreies Druckbild.
- Vergessen Sie nicht, auf dem Deckblatt die Angaben zur Person 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, 12. Februar 2013 .....
(Unterschrift)

Ich bin damit einverstanden, dass mein Prüfungsergebnis unter Angabe der Matrikelnummer anonymisiert veröffentlicht wird:

ja: ☐ nein: ☐

Erlangen, 12. Februar 2013 .....
(Unterschrift)

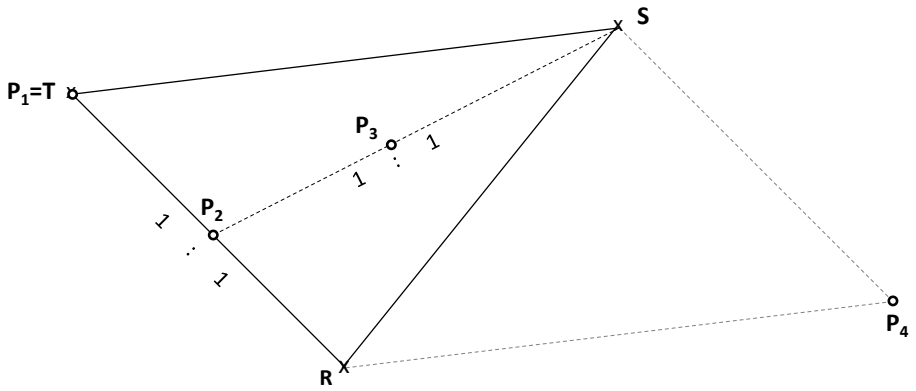
1 Komplexität (4 Punkte)

Bestimmen Sie die Komplexität der folgenden Operationen. Dabei sollen Sie annehmen, dass die Länge der Spaltenvektoren n und die Größe von Matrizen $n \times n$ ist.

Lösen von $\mathbf{A}\vec{x} = \vec{b}$, für eine gegebene QR-Zerlegung von $\mathbf{A}$	$\mathbf{O}$ ( )
Bestimmung der LU-Zerlegung von $\mathbf{A}$	$\mathbf{O}$ ( )
Bestimmung der Hauptachsentransformation (PCA) von n Datenpunkten im $\mathbb{R}^3$	$\mathbf{O}$ ( )
Matrix-Vektor Multiplikation	$\mathbf{O}$ ( )
Matrix-Skalar Multiplikation	$\mathbf{O}$ ( )
Bestimmung der Koeffizienten des Interpolationspolynoms bzgl. der NEWTON-Basis (n Datenpunkte)	$\mathbf{O}$ ( )
Midpoint subdivision einer Bézierkurve vom Grad n	$\mathbf{O}$ ( ) ‘
Bestimmung der Determinante einer Matrix bei gegebener LU-Zerlegung	$\mathbf{O}$ ( )

2 Baryzentrische Koordinaten (9 Punkte)

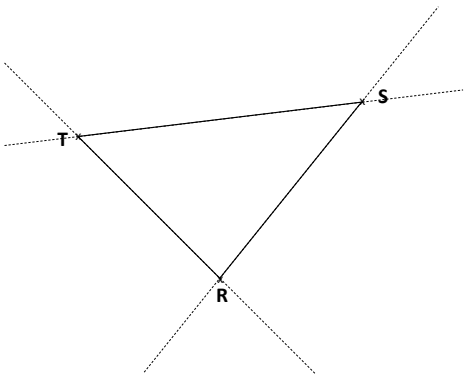
Gegeben ist das Dreieck $\triangle(RST)$ und vier Punkte $P_i,\ i = 1, 2, 3, 4$. *Hinweis:* In den folgenden Aufgaben beziehen sich die baryzentrischen Koordinaten ρ auf R, σ auf S und τ auf T.



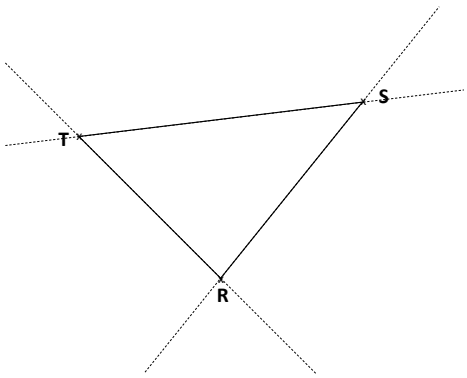
a) Bestimmen Sie die baryzentrischen Koordinaten $(\rho_i, \sigma_i, \tau_i)$ der Punkte P_i bzgl. des Dreiecks $\triangle(RST)$.

b) Schraffieren Sie in den nachfolgenden Abbildungen jeweils den spezifizierten Bereich

$\rho < 0\ ,\quad \sigma > 0\ ,\quad \tau > 0$



$\rho > 0\ ,\quad \sigma < 0\ ,\quad \tau < 0$



c) In den Eckpunkten des Rechtecks $\overline{ABCD}$ mit $A = [2, 2]$, $B = [10, 2]$, $C = [10, 6]$, $D = [2, 6]$ sind folgende Werte gegeben:

$f_A = 12$, $f_B = 8$, $f_C = 16$, $f_D = 12$.

Berechnen Sie mittels bilinear Interpolation den Wert an der Stelle $P = [8, 4]$.

d) Angaben wie in Teil c).
Berechnen Sie nun den Wert an der Stelle P indem Sie im Dreieck $\triangle(BCD)$ die Werte f_B , f_C , f_D linear interpolieren.

3 Falten und Filtern (11 Punkte)

a) Welche der folgenden 2D-Filter sind separierbar? Geben Sie im Falle der Separierbarkeit jeweils auch die zugehörigen 1D-Filter an!

$$\frac{1}{4}\begin{bmatrix}0 & 1 & 0\\ 1 & 0 & 1\\ 0 & 1 & 0\end{bmatrix}$$

$$\begin{bmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{bmatrix}$$

$$\frac{1}{8}\begin{bmatrix}1 & 1 & 0 & 0\\ 0 & 0 & 1 & 1\\ 1 & 1 & 0 & 0\\ 0 & 0 & 1 & 1\end{bmatrix}$$

$$\frac{1}{12}\begin{bmatrix}1 & 1 & 1 & 1\\ 1 & 1 & 1 & 1\\ 1 & 1 & 1 & 1\end{bmatrix}$$

b) Ein (Grau-)bild mit $N \times N$ Pixeln soll mit einem nicht separierbaren Filter der Größe 5×5 gefiltert werden. Dabei werden nur die inneren $(N-2) \times (N-2)$ Pixel berechnet.

Bestimmen Sie den Rechaufwand (Anzahl der Multiplikationen).

c) Nun wird das Bild von Teil b) mit einem separierbaren Filter der Größe 5×5 gefiltert. Bestimmen sie wiederum den Rechaufwand (Anzahl der Multiplikationen) unter Ausnutzung der Separierbarkeit.

d) Bestimmen Sie eine mögliche 3×3 Filtermaske des diskreten `Laplace`-Operators im $\mathbb{R}^2$.

e) Beschreiben Sie mit je einem Stichwort, welche Auswirkungen ein Sobel-Filter und ein Tiefpassfilter auf ein Graustufenbild haben.

4 QR-Zerlegung (7 Punkte)

Gegeben ist die QR-Zerlegung der Matrix $\mathbf{A}$ mit

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

a) Lösen Sie unter Verwendung der QR-Zerlegung das Gleichungssystem $\mathbf{A}\vec{x} = \vec{b}$ für die rechte Seite $\vec{b} = [-7, -8, 2, -9]^T$:

b) Bestimmen Sie den Betrag der Determinante: $|\det(A)|$.

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

$$\mathbf{B} = \begin{bmatrix} 3 & -1 \\ -4 & -1 \end{bmatrix}$$

5 Nichtlineare Optimierung (8 Punkte)

Betrachten Sie die Matrix

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

und das quadratische Funktional

$$Q(\vec{x}) = \vec{x}^T \mathbf{A} \vec{x} + 2\vec{b}^T \vec{x} + \gamma, \qquad \vec{b} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \ \gamma = 1 \ .$$

a) Bestimmen Sie eine zu $\begin{bmatrix} 0 \\ -1 \end{bmatrix}$ A-konjugierte Richtung.

b) Führen Sie zwei Schritte des cg-Verfahrens zur Bestimmung des Minimums von Q durch.

Beginnen Sie im Punkt $\vec{x}_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ mit der Suchrichtung $\vec{s}_0 = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$ und verwenden Sie die optimale Schrittweite

$t_i := -\frac{(\mathbf{A}\vec{x}_i + \vec{b})^T \vec{s}_i}{\vec{s}_i{}^T \mathbf{A} \vec{s}_i}$ für die Suchrichtung $\vec{s}_i$.

Zur Erinnerung:

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

c) Führen Sie einen Schritt des Gradienten-Verfahrens zur Bestimmung des Minimums von Q durch.
Beginnen Sie wiederum im Punkt $\vec{x}_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ und verwenden Sie die optimale Schrittweite
$t_i := -\frac{(\mathbf{A}\vec{x}_i + \vec{b})^T \vec{s}_i}{\vec{s}_i{}^T \mathbf{A} \vec{s}_i}$ für die Suchrichtung $\vec{s}_i$.

d) Das Gradienten-Verfahren hat Konvergenzordnung 1 (lineare Konvergenz).
Erklären Sie für eine konvergente Folge (x_i) mit Grenzwert x^* diesen Begriff.

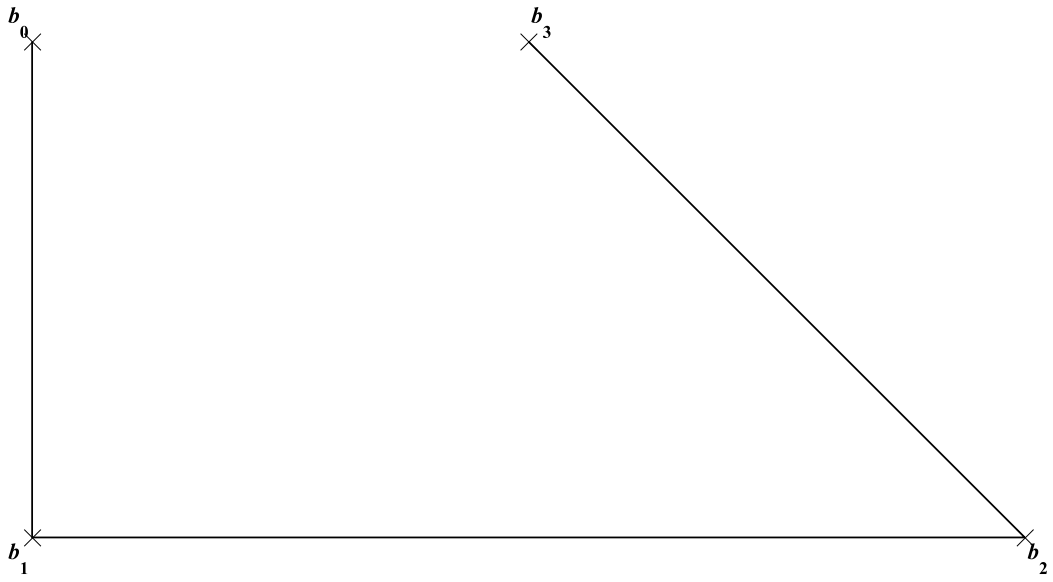
6 Bézier-Kurven (7 Punkte)

a) Werten Sie mit Hilfe des Algorithmus von DE CASTELJAU die BÉZIER-Kurve, definiert auf dem Intervall $[0, 1]$ mit den Kontrollpunkten

$$\vec{b}_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \vec{b}_1 = \begin{bmatrix} 9 \\ 0 \end{bmatrix} \quad \vec{b}_2 = \begin{bmatrix} 9 \\ 6 \end{bmatrix} \quad \vec{b}_3 = \begin{bmatrix} 0 \\ 9 \end{bmatrix},$$

an der Stelle $t_0 = \frac{1}{3}$ aus (rechnerisch).

b) Werten Sie mit Hilfe des Algorithmus von DE CASTELJAU die folgende BÉZIER-Kurve geometrisch an der Stelle $t_1 = \frac{2}{3}$ aus.



c) Beweisen Sie die Endpunktinterpolation von BÉZIER-Kurven. Das heisst;
Für jede BÉZIER-Kurve $\vec{f}(u) = \sum_{i=0}^{n} \vec{b}_i B_i^n(u), u \in [0,1]$ muss gelten $\vec{f}(0) = \vec{b}_0$ und $\vec{f}(1) = \vec{b}_n$.

7 Singulärwertzerlegung und PCA (9 Punkte)

Gegeben sei die Singulärwertzerlegung (SVD) der Matrix $\mathbf{A} = \mathbf{USV}^T$:

$$\underbrace{\frac{1}{24}\begin{bmatrix}-16 & -8 & -4 & 4\\ -8 & -4 & -20 & -16\\ -11 & -19 & -5 & -13\end{bmatrix}}_{\mathbf{A}} = \underbrace{\frac{1}{3}\begin{bmatrix}-1 & -2 & -2\\ -2 & 2 & -1\\ -2 & -1 & 2\end{bmatrix}}_{\mathbf{U}} \cdot \underbrace{\begin{bmatrix}\frac{6}{4} & 0 & 0 & 0\\ 0 & \frac{3}{4} & 0 & 0\\ 0 & 0 & \frac{2}{4} & 0\end{bmatrix}}_{\mathbf{S}} \cdot \underbrace{\frac{1}{2}\begin{bmatrix}1 & 1 & 1 & 1\\ 1 & 1 & -1 & -1\\ 1 & -1 & 1 & -1\\ 1 & -1 & -1 & 1\end{bmatrix}^T}_{\mathbf{V}^T} .$$

a) Bestimmen Sie für die Matrix $\mathbf{A}$:

- die Singulärwerte:

- den Rang:

- den Bildraum (welche Vektoren spannen den Bildraum auf?):

- den Kern (welche Vektoren spannen den Kern auf?):

- und die Konditionszahl bzgl. der EUKLIDISCHEN-Norm $|||\cdot|||_2$:

b) Gegeben sind folgende Datenpunkte $P_i \in \mathbb{R}^2$:

$$P_0 = \begin{bmatrix} 4 \\ 0 \end{bmatrix},\ P_1 = \begin{bmatrix} -1 \\ -1 \end{bmatrix},\ P_2 = \begin{bmatrix} 1 \\ 2 \end{bmatrix},\ P_3 = \begin{bmatrix} -1 \\ 2 \end{bmatrix},\ P_4 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}\ .$$

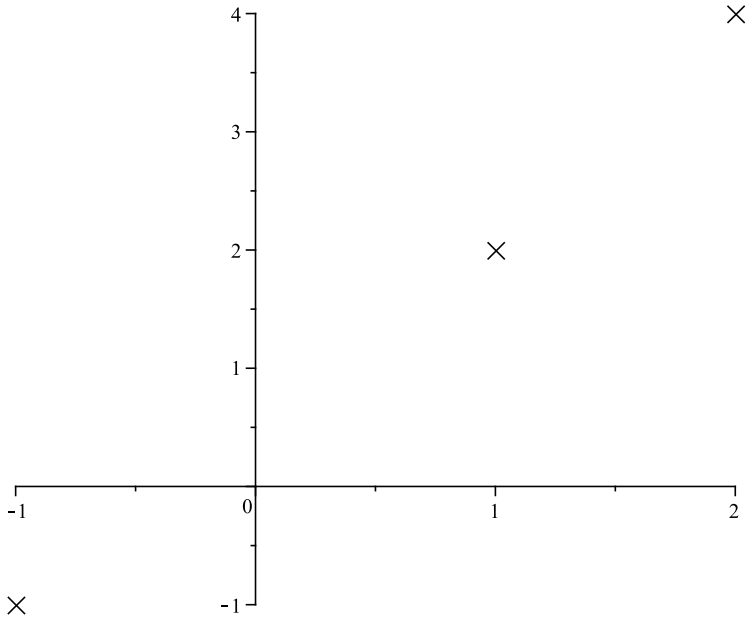
Bestimmen Sie die Kovarianzmatrix und führen Sie die Hauptachsentransformation (PCA) für die gegebenen Datenpunkte P_i durch (bestimmen Sie also die zugehörigen Hauptachsen).

8 Polynominterpolation (10 Punkte)

Gegeben seien folgende Punkte:

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

a) Skizzieren Sie in der folgenden Abbildung den nearest neighbor Interpolanten zu den obigen Daten.



b) Berechnen Sie die Funktion $l(x) : [-1, 2] \mapsto \mathbb{R}$, welche obige Werte stückweise linear interpoliert.

Zur Erinnerung:

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

Im Weiteren sollen die obigen Daten durch ein Polynom interpoliert werden.

c) Bestimmen Sie die LAGRANGE-Polynome für die obigen Stützstellen $\{-1, 1, 2\}$.

$l_0(x) =$

$l_1(x) =$

$l_2(x) =$

Bestimmen Sie des Weiteren die Koeffizienten des Interpolationspolynoms bzgl. der LAGRANGE-Basis.

d) Bestimmen Sie NEWTON-Polynome für die obigen Stützstellen $\{-1, 1, 2\}$.

$n_0(x) =$

$n_1(x) =$

$n_2(x) =$

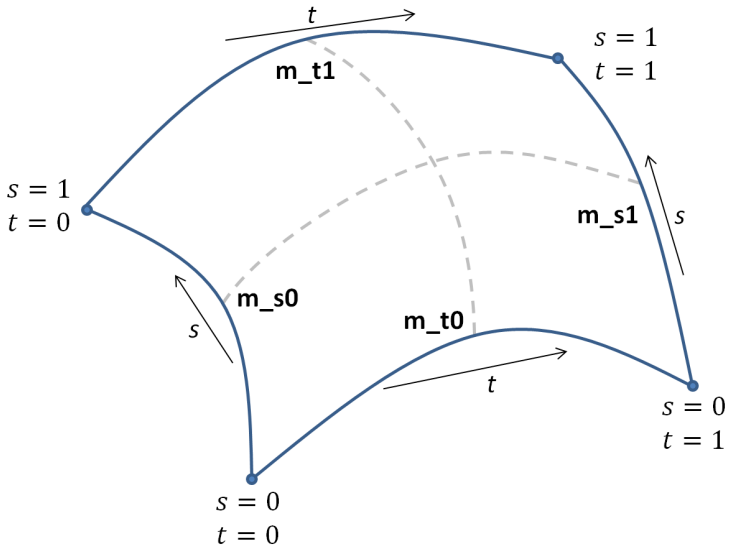
Bestimmen Sie des Weiteren die Koeffizienten des Interpolationspolynoms bzgl. der NEWTON-Basis.
Hinweis: Aitken-Neville!

9 Programmierung: Coons Patches (10 Punkte)

In dieser Aufgabe soll ein Coons Patch, repräsentiert durch die Klasse `CoonsPatch`, implementiert werden. Verwenden Sie dafür `C++` Syntax. Entgegen der Übungen ist **keine** Fehlerbehandlung erforderlich.

```
class CoonsPatch {
 public:
    ...
    //! Wertet den Patch an (s, t) aus
    float evaluateAt(float s, float t) const;

    //! Berechnet die Ableitung des Patches in s-Richtung an der Stelle (s, t)
    float evaluateDerivativeS(float s, float t) const;

 private:

    //! Funktionen, die den Patch definieren (siehe Grafik!)
    const ParametricFunction &m_s0;   //linke Funktion; hat den Parameter s
    const ParametricFunction &m_s1;   //rechte Funktion; hat den Parameter s
    const ParametricFunction &m_t0;   //untere Funktion; hat den Parameter t
    const ParametricFunction &m_t1;   //obere Funktion; hat den Parameter t
};

class ParametricFunction {
 public:
    //! Wertet die Funktion an der Stelle t (in [0,1]) aus
    virtual float f(float t) const = 0;

    //! Auswertung der Ableitung der Funktion an der Stelle t (in [0,1])
    virtual float d(float t) const = 0;
};
```

Visualisierung des Coons Patches mit entsprechender Zuordnung (und Parametrisierung) der Randkurven:



a) Implementieren Sie die Methode `evaluateAt(float s, float t)`, welche das Coons Patch an der Stelle (s, t) auswertet. Sie können davon ausgehen, dass sowohl s als auch t im Intervall $[0, 1]$ liegen. Geben Sie anschließend den berechneten Wert zurück. Achten Sie dabei auf die korrekte Zuordnung der Randkurven (siehe Grafik!).

```
float CoonsPatch::evaluateAt(float s, float t) const {

}
```

b) Implementieren Sie die Methode `evaluateDerivativeS(float s, float t)`, welche die Ableitung des Coons Patches in s-Richtung an der Stelle (s, t) auswertet. Sie können davon ausgehen, dass sowohl s als auch t im Intervall $[0, 1]$ liegen. Geben Sie anschließend den berechneten Wert zurück. Achten Sie dabei auf die korrekte Zuordnung der Randkurven (siehe Grafik!).

```
float CoonsPatch::evaluateDerivativeS(float s, float t) const {

}
```

10 Dünnbesetzte Matrizen (15 Punkte)

Die Klasse `CCSMatrix` speichert dünn besetzte Matrizen im `Compressed-Column-Storage`-Format; die Klasse `Matrix` voll besetzte Matrizen. In dieser Aufgabe sollen Sie ausgewählte Methoden beider Klassen implementiert werden (verwenden Sie `C++` Syntax). Entgegen der Übung ist **keine** Fehlerbehandlung erforderlich!

```
class CCSMatrix {
        //! Gibt der Matrixklasse Zugriff auf die privaten Variablen
        friend class Matrix;
    public:
        //! Erstellt eine leere CCS-Matrix (nur 0 Elemente)
        CCSMatrix(unsigned int height, unsigned int width);

        //! Erstellt eine CCS-Matrix aus einer normalen Matrix
        CCSMatrix(Matrix &other);

        //! Destruktor: Gibt den Speicher frei
        ~CCSMatrix();

        //! Liefert den i,j-ten Eintrag der Matrix (i ist Zeilen-, j ist Spaltenindex)
        float getEntry(unsigned int i, unsigned int j);
    private:
        unsigned int _height;                   //! Hoehe der Matrix
        unsigned int _width;                    //! Breite der Matrix
        unsigned int _nonZeroElements;          //! Anzahl der nicht-0-Elemente in der Matrix
        float* _values;                         //! Wertearray
        unsigned int* _rowIndices;              //! Zeilenindexarray
        unsigned int* _colPtr;                  //! Spaltenpointer
};

class Matrix {
    public:
        //! Multiplikation mit einer CCS-Matrix: Gibt die Ergebnismatrix zurueck
        Matrix operator*(CCSMatrix &other);

        //! Akzessor- und Mutatorfunktionen (i ist Zeilen-, j ist Spaltenindex)
        float getEntry(unsigned int i, unsigned int j);
        void setEntry(unsigned int i, unsigned int j, float value);
        float& operator()(unsigned int i, unsigned int j);

        unsigned int getHeight();               //! Liefert die Hoehe der Matrix
        unsigned int getWidth();                //! Liefert die Breite der Matrix
        unsigned int getNumNonZeroEntries();    //! Liefert die Anzahl der nicht-0 Werte
    private:
        unsigned int _height;                   //! Hoehe der Matrix
        unsigned int _width;                    //! Breite der Matrix
        float* values;                          //! Werte der Matrix
};
```

Hinweis: Beachten Sie, dass Indizierungen stets bei 0 beginnen!

Bitte wenden!

a) Implementieren Sie den Konstruktor `CCSMatrix(unsigned int height, unsigned int width)`, welcher eine leere CCS-Matrix erstellt. Das bedeutet, dass alle Elemente der Matrix 0 sind. Achten Sie darauf, dass der Inhalt der privaten Membervariablen dem Zustand des Objektes bzw. der Matrix entspricht.

```
CCSMatrix::CCSMatrix(unsigned int height, unsigned int width) {









}
```

b) Implementieren Sie die Methode `float getEntry(unsigned int i, unsigned int j)` der Klasse `CCSMatrix`, welche den (i, j)-ten Eintrag der Matrix zurück liefert (i ist Zeilen-, j ist Spaltenindex).

```
float CCSMatrix::getEntry(unsigned int i, unsigned int j) {









}
```

c) Implementieren Sie den Konstruktor `CCSMatrix(Matrix &other)`, welcher eine CCS-Matrix aus einer normalen Matrix erstellt. Es dürfen natürlich keine 0-Werte gespeichert werden. Achten Sie darauf, dass der Inhalt der privaten Membervariablen dem Zustand des Objektes bzw. der Matrix entspricht.

```
CCSMatrix::CCSMatrix(Matrix &other) {




}
```

d) Überladen Sie den Multiplikations-Operator, welcher eine normale `Matrix` mit einer `CCSMatrix` multipliziert. Nutzen Sie dabei das CCS-Format effizient aus.

```
Matrix Matrix::operator*(CCSMatrix &other) {
    Matrix result(_height, other._width);

    return result;
}
```