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

23. Juli 2012

Algorithmik kontinuierlicher Systeme — 23. Juli 2012

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	6	9	12	9	10	15	8	6	11
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 (30 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, 23. Juli 2012 .....
(Unterschrift)

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

ja: ☐ nein: ☐

Erlangen, 23. Juli 2012 .....
(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 Singulärwertzerlegung von $\mathbf{A}$	$\mathbf{O}$ ( )
Bestimmung der LU-Zerlegung der k-diagonalen Matrix $\mathbf{A}$	$\mathbf{O}$ ( )
Berechnung der Inversen einer Diagonalmatrix $\mathbf{A}$ mit vollem Rang	$\mathbf{O}$ ( )
Matrix-Matrix Multiplikation	$\mathbf{O}$ ( )
Anzahl der Schritte zum Lösen von $\mathbf{A}x = b$ mittels CG-Verfahren	$\mathbf{O}$ ( )
Berechnung des Winkels zwischen zwei Vektoren	$\mathbf{O}$ ( )
Auswerten einer Bézierkurve vom Grad n mittels De-Casteljau	$\mathbf{O}$ ( )
Exponentieren (Exponent k) einer Matrix $\mathbf{A}$ unter Ausnutzung der Diagonalisierung	$\mathbf{O}$ ( )

2 Lineare Gleichungssysteme (6 Punkte)

Gegeben sei die Matrix $\mathbf{A}$ mit

$$\mathbf{A} = \begin{bmatrix} 3 & 3 & 0 \\ 12 & 11 & 1 \\ 9 & 10 & 1 \end{bmatrix}.$$

a) Bestimmen Sie die $\mathbf{LU}$-Zerlegung (ohne Pivotisierung) von $\mathbf{A}$.

b) Gegeben ist die **QR**-Zerlegung der Matrix **B** mit

$$\mathbf{B} = \begin{bmatrix} 2/3 & 0 & 1/3 & -2/3 \\ 2/3 & 0 & -2/3 & 1/3 \\ 1/3 & 0 & 2/3 & 2/3 \\ 0 & 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 3 & -12 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}.$$

Lösen Sie $\min_{\vec{x}} \|\mathbf{B}\vec{x} - \vec{b}\|_2$ mit Hilfe der **QR**-Zerlegung für $b = [2, -2, 3, -1]^T$.

3 SVD (9 Punkte)

a) Gegeben ist die Matrix $\mathbf{A}$ mit

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

Gegeben sind außerdem die Eigenwerte von $\mathbf{A}^T\mathbf{A}$ mit $\lambda_{1,2,3} = 4, 1, \frac{1}{4}$, sowie die zugehörigen Eigenvektoren von $\mathbf{A}^T\mathbf{A}$ (selbe Reihenfolge wie λ_i)

$$\vec{e}_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \qquad \vec{e}_2 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \qquad \vec{e}_3 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}.$$

Bestimmen Sie die Singulärwertzerlegung von $\mathbf{A}$. D.h. berechnen Sie $\mathbf{U}$, $\mathbf{\Sigma}$, $\mathbf{V}$.

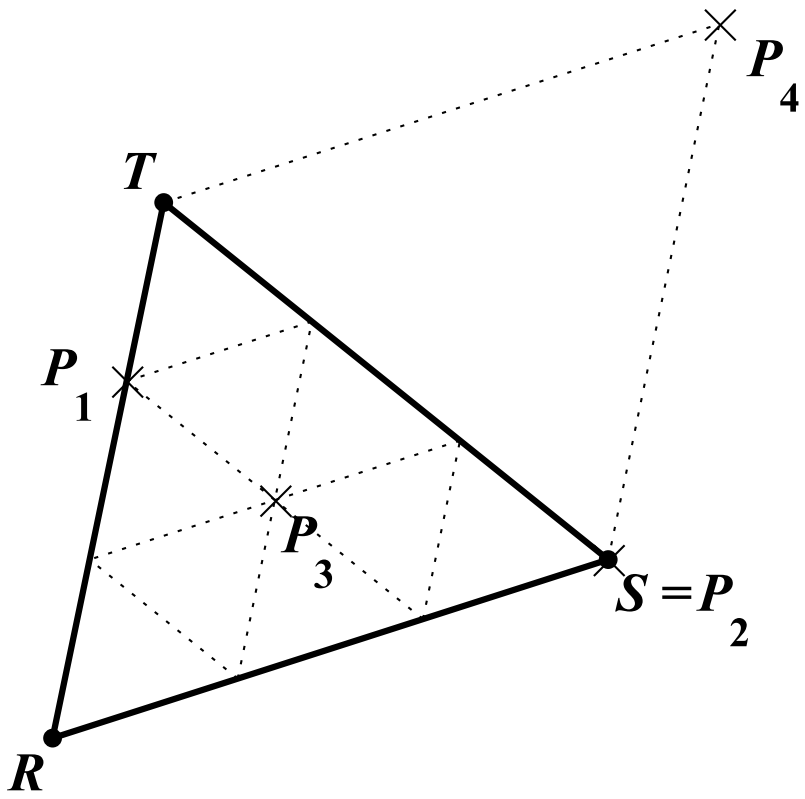
b) Gegeben ist die Singulärwertzerlegung der Matrix $\mathbf{B}$:

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

Bestimmen Sie die Norm $|||\mathbf{B}|||_2$ und die Konditionszahl $\kappa_2(\mathbf{B})$, sowie die beste Rang-1-Approximation im Sinne der Frobenius Norm von $\mathbf{B}$.

4 Baryzentrische Koordinaten (12 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) Markieren Sie in der obigen Abbildung den Bereich in dem $\tau < 0 \wedge \sigma < 0$.

c) Betrachten Sie das Dreieck $\triangle(ABC)$ mit $A = [7, 1]$, $B = [7, 4]$, $C = [1, 4]$ und berechnen Sie die baryzentrischen Koordinaten (α, β, γ) bzgl. $\triangle(ABC)$ des Punktes Q mit $Q = [6, 3]$.

d) Gegeben ist wiederum das Dreieck $\triangle(ABC)$ von Teil c), sowie der Punkt $X(\alpha, \beta, \gamma)$ der bezüglich des Dreieckes baryzentrisch interpoliert wird. Geben Sie den Definitionsbereich der baryzentrischen Koordinaten (α, β, γ) an, sodass $X(\alpha, \beta, \gamma)$ auf der Strecke $\overline{BC}$ liegt.

e) In den Eckpunkten des Dreiecks $\triangle(ABC)$ von Teil c) sind die Werte gegeben:

$$f_A = -1 \ , \qquad f_B = 1 \ , \qquad f_C = 3 \ .$$

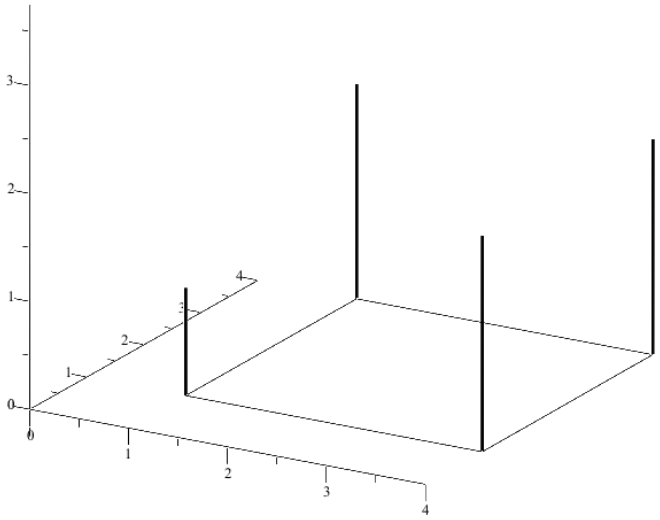
Diese Werte werden linear ins Innere interpoliert. Bestimmen Sie den Wert f_M des Interpolanten im Mittelpunkt $M = \frac{1}{3}(A + B + C)$.

f) Im Quadrat $[x_0, x_1]^2$ ($x_0 = 1$, $x_1 = 4$) sind in den Eckpunkten die folgenden Werte gegeben:

$$f(x_0, x_0) = f_{00} = 1 \ , \quad f(x_1, x_0) = f_{10} = 2 \ , \quad f(x_1, x_1) = f_{11} = 2 \ , \quad f(x_0, x_1) = f_{01} = 2 \ .$$

Diese werden ins Innere bilinear interpoliert.

Skizzieren Sie in der nachfolgenden Abbildung die vier Randkurven des bilinearen Interpolanten.



5 Bézier-Kurven (9 Punkte)

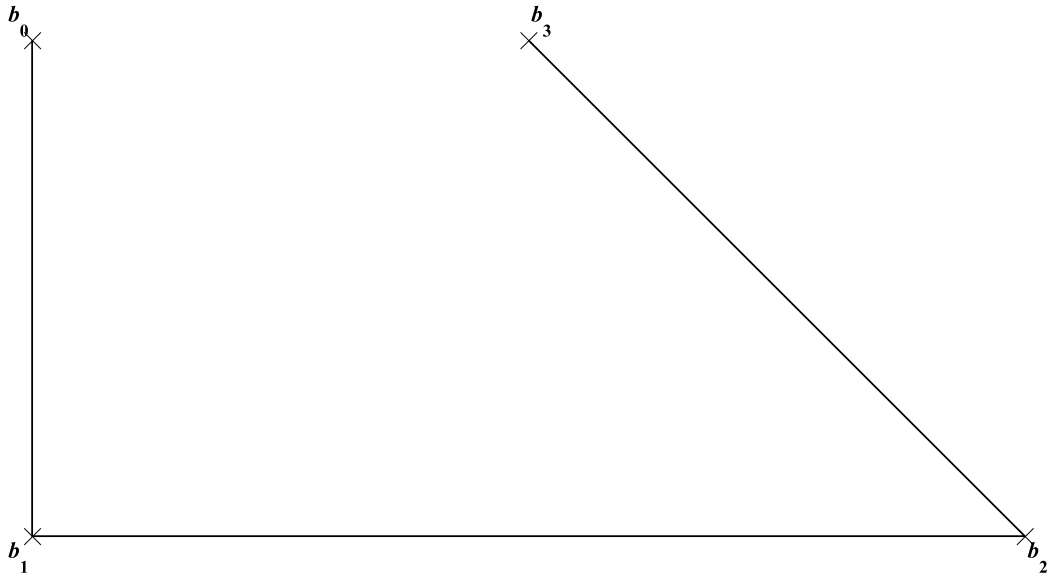
a) Betrachten Sie die BÉZIER-Kurve $C(t)\,,\ (0 \leq t \leq 1)$ mit den Kontrollpunkten

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

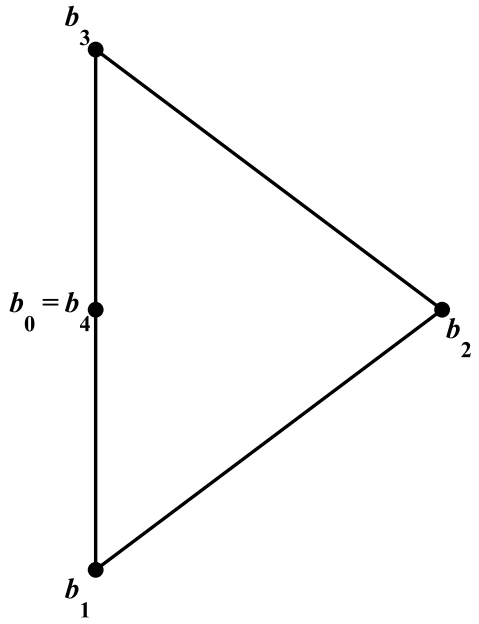
Führen Sie für die Kurve $C(t)$ einen *midpoint subdivision* Schritt durch und berechnen Sie die Kontrollpunkte $\vec{c}_0$, $\vec{c}_1$, $\vec{c}_2$ und $\vec{c}_3$ der linken Teilkurve, sowie die Kontrollpunkte $\vec{d}_0$, $\vec{d}_1$, $\vec{d}_2$ und $\vec{d}_3$ der rechten Teilkurve.

b) Bestimmen Sie die *midpoint subdivision* einer BÉZIER-Kurve geometrisch.

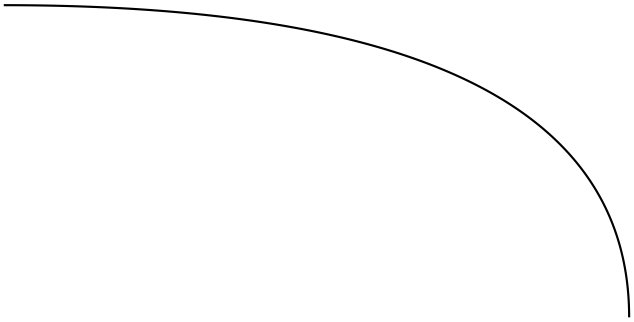
Ergänzen Sie dazu die nachfolgende Abbildung und benennen Sie die Kontrollpunkte $\vec{c}_0$, ... $\vec{c}_3$ der linken Teilkurve, sowie die Kontrollpunkte $\vec{d}_0$, ..., $\vec{d}_3$ der rechten Teilkurve.



c) Skizzieren Sie die Bézier-Kurve zu folgendem Kontrollpolygon



d) Die nachfolgende Abbildung zeigt eine quadratische Bézier-Kurve.
Zeichnen Sie in diese Abbildung die Kontrollpunkte dieser Kurve und ihr Kontrollpolygon.

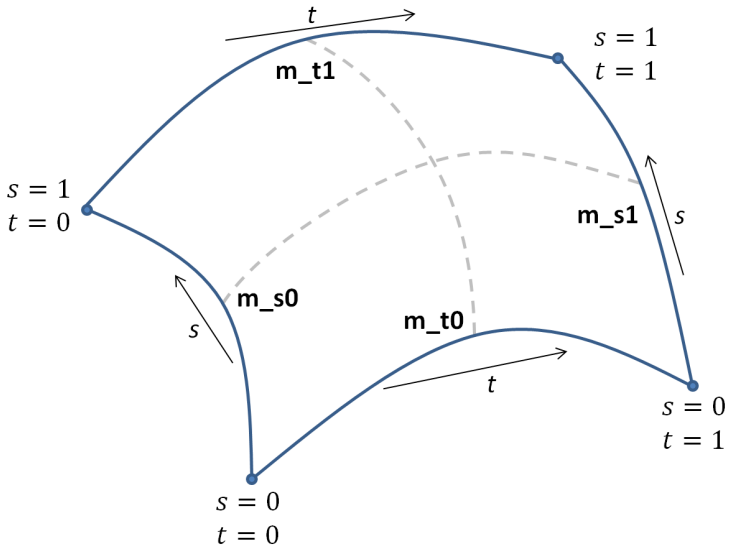


6 Programmierung: Coons Patches (10 Punkte)

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 {

}
```

7 Programmierung: Nichtlineare Optimierung (15 Punkte)

In dieser Aufgabe geht es um nichtlineare Optimierung einer Zielfunktion $f(\vec{x})$. Achtung: Entgegen der Aufgabenstellung der Übung wird in dieser Aufgabe ein **Maximum** der Funktion gesucht!

Verwenden Sie für die Bearbeitung der Teilaufgaben `C++`-Syntax. Entgegen der Übungen ist **keine** Fehlerbehandlung erforderlich.

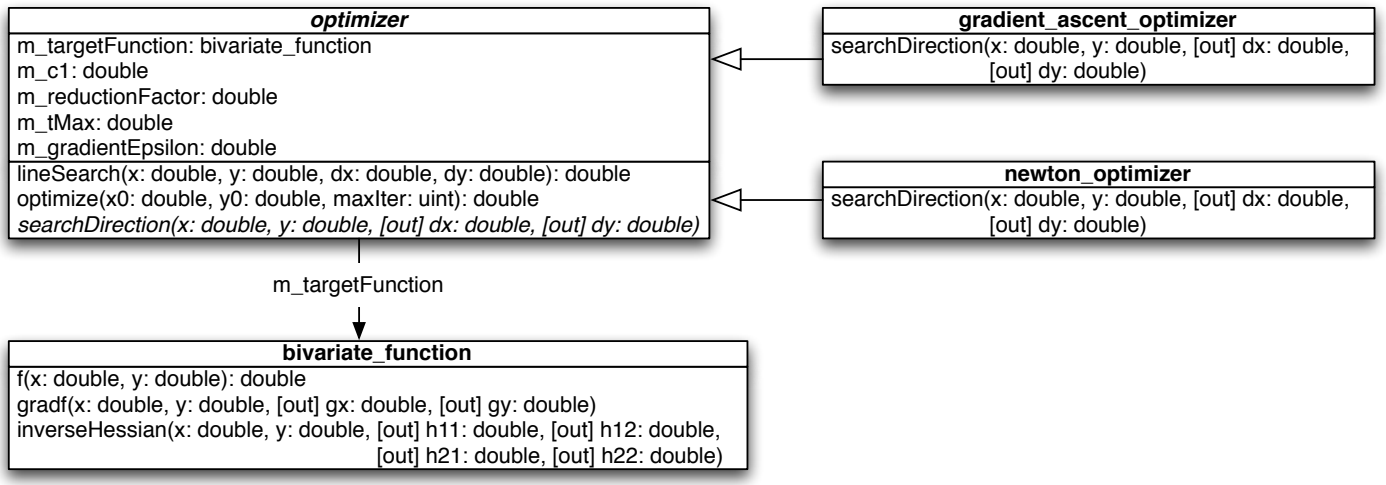
Um Ihnen die Implementierung zu erleichtern, werden folgende vereinfachende Annahmen gemacht:

- Es werden nur bivariate Funktionen $f(x, y)$ optimiert
- Wir suchen immer ein **Maximum** der Zielfunktion: $(x^*, y^*) = \underset{(x,y)}{\operatorname{argmax}} f(x, y)$

Ein allgemeiner Optimierungsalgorithmus kann folgende Form haben:

1. Der Startpunkt sei $\vec{x}_0$
2. Für $k = 0, \ldots, \text{maxIter} - 1$:
   1. Bestimme eine Suchrichtung $\vec{d}_k$
   2. Finde die optimale Schrittweite: $t^* = \underset{t}{\operatorname{argmax}} f(\vec{x}_k + t \cdot \vec{d}_k)$
   3. Neue Schätzung für die gesuchte Stelle: $\vec{x}_{k+1} = \vec{x}_k + t^* \cdot \vec{d}_k$
   4. Schleifenabbruch bei Konvergenz

Im Folgenden sollen die einzelnen Schritte des Optimierungsalgorithmus implementiert werden. Zur Bestimmung der Suchrichtung sollen zwei Möglichkeiten untersucht werden: Gradientenaufstieg (*gradient ascent*) und das Newton-Verfahren. Hier ist ein Überblick über die verwendeten Klassen:

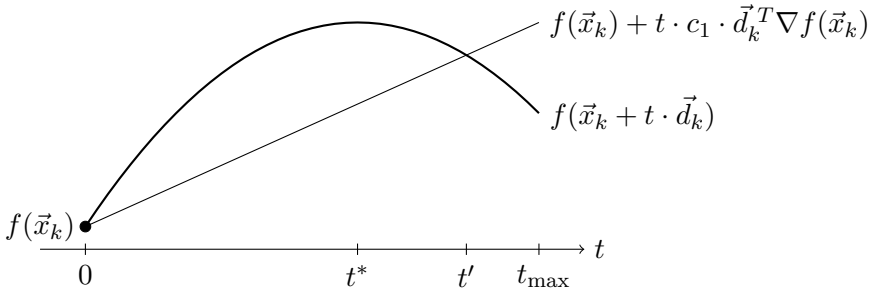


```
class optimizer {
 public:
     ...

    //! Backtracking line search von Position (x, y) in Richtung (dx, dy).
    //! Hier werden die Instanzvariablen m_c1, m_tMax und m_reductionFactor benoetigt
    virtual double lineSearch(const double x, const double y,
                              const double dx, const double dy) const;

    //! Findet die Suchrichtung; die Implementierung erfolgt in den Unterklassen
    virtual void searchDirection(const double x, const double y, double &dx, double &dy) const = 0;

    //! Optimiert die Zielfunktion mit dem Startwert (x0, y0)
    //! Gibt den Funktionswert am gefundenen Optimum zurueck
    double optimize(double x0, double y0, const unsigned int maxIter);

 protected:
    const bivariate_function &m_targetFunction;

    const double m_c1;
    const double m_reductionFactor;
    const double m_tMax;
    double m_gradientEpsilon;
};



class bivariate_function {    //Interface einer bivariaten Funktion
 public:
    bivariate_function() {}
    virtual ~bivariate_function() {}

    //! Auswertung der Funktion f(x, y)
    virtual double f(const double x, const double y) const = 0;

    //! Auswertung des Gradienten grad(f(x, y)); Rueckgabewerte sind gx und gy
    virtual void gradf(const double x, const double y, double &gx, double &gy) const = 0;

    //! Auswertung der inversen Hesse-Matrix (f(x, y)); die Matrix hat die Form
    //!          [h11, h12]
    //! H^{-1} = [        ]
    //!          [h21, h22]
    virtual void inverseHessian(const double x, const double y,
                          double &h11, double &h12, double &h21, double &h22) const;
};
```

a) Setzen Sie zunächst den allgemeinen (oben beschriebenen) Optimierungsalgorithmus um, indem Sie die Methode `optimize(...)` implementieren. Verwenden Sie dazu die Methode `lineSearch(...)` und die abstrakte Methode `searchDirection(...)` der Klasse `optimizer`, die in den folgenden Teilaufgaben implementiert werden. Konvergenz tritt ein, sobald die 2-Norm des Gradienten eine vorgegebene Schwelle ε unterschreitet (ε ist durch die Membervariable `m_gradientEpsilon` gegeben). Den Funktionswert und den Gradienten der Zielfunktion an einer bestimmten Stelle können Sie mit Hilfe der Instanzvariable `m_targetFunction` bestimmen (siehe Klasse `bivariate_function`). Geben Sie anschließend das berechnete Optimum als Returnwert zurück.

```
double optimizer::optimize(double x0, double y0, const unsigned int maxIter) {


}
```

b) Um in Schritt 2.2 der oben beschriebenen Optimierung die Schrittweite zu bestimmen, soll eine *backtracking line search* implementiert werden. Hierbei wird nicht die exakte Lösung des eindimensionalen Optimierungsproblems $t^* = \text{argmax}_t f(\vec{x}_k + t \cdot \vec{d}_k)$ gefunden (siehe Zeichnung), sondern die Stelle t' approximiert, wo die Gerade $f(\vec{x}_k) + t \cdot c_1 \cdot \vec{d}_k^{\,T} \nabla f(\vec{x}_k)$ die Zielfunktion überschreitet.



Dabei ist $c_1 \in (0, 1)$ eine frei wählbare Konstante. Um die Stelle t' zu finden, beginnt man die Suche bei einem gegebenen $t_{\max}$ und geht „rückwärts“, solange die Gerade einen größeren Funktionswert als die Zielfunktion hat. Bei jedem Schritt wird das aktuelle t um einen Faktor $\tau < 1$ verringert: $t_{i+1} = \tau \cdot t_i$ (τ ist durch die Membervariable `m_reductionFactor` gegeben). Implementieren Sie dies in der Methode `lineSearch(...)`.

```
double optimizer::lineSearch(const double x, const double y,
                             const double dx, const double dy) const {
```

```
}
```

c) Implementieren Sie die Methode `gradient_ascent_optimizer::searchDirection(...)`. In dieser soll die Suchrichtung entsprechend dem Gradientenaufstiegsverfahren gewählt werden.

```
void gradient_ascent_optimizer::searchDirection(double x, const double y,
                                                double &dx, double &dy) const {









}
```

d) Das Newton-Verfahren approximiert die zu maximierende Funktion an jeder Stelle durch eine quadratische Funktion und wählt als Suchrichtung $\vec{d_k}$ den Schritt zum Maximum dieser quadratischen Funktion. Weil zusätzlich zum Gradienten $\nabla f(\vec{x}_k)$ die Information über die Krümmung der Funktion in Form der Hesse-Matrix $\boldsymbol{H}_f(\vec{x}_k)$ verwendet wird, kann so eine schnellere Konvergenz erreicht werden. Die neue Suchrichtung für das Newton-Verfahren ist $\vec{d_k} = -\boldsymbol{H}_f(\vec{x}_k)^{-1}\ \nabla f(\vec{x}_k)$. Prüfen Sie, ob $\vec{d_k}^T \nabla f(\vec{x}_k) > 0$, falls nicht, wechseln Sie das Vorzeichen von $\vec{d_k}$. Implementieren Sie dazu die Methode `newton_optimizer::searchDirection(...)`. Die Klasse `bivariate_function` bietet eine Methode `inverseHessian(...)`, die die Komponenten der inversen Hesse-Matrix zurückliefert.

```
void newton_optimizer::searchDirection(const double x, const double y,
                                       double &dx, double &dy) const {
```

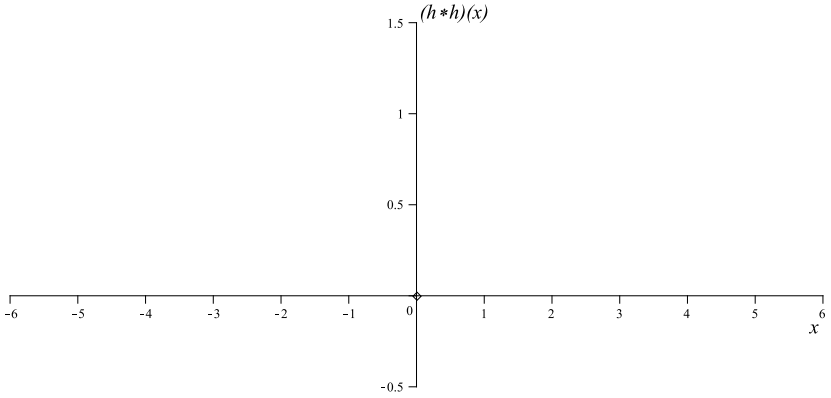
```
}
```

8 Falten und Filtern (8 Punkte)

a) Betrachten Sie zunächst die Funktion

$$h(x) = \begin{cases} 0.5 & \text{falls} \quad -1 \leq x \leq 1 \\ 0 & \text{sonst} \end{cases}$$

Bestimmen Sie zeichnerisch die Faltung $h * h$.



b) Betrachten Sie nun die Funktion

$$g(x) = \begin{cases} 1 & \text{falls} \quad 0 \leq x \leq 1 \\ 0 & \text{sonst} \end{cases}$$

Berechnen Sie die Faltung $(g * g)(x)$.
Hinweis: $(f * g)(x) = \int f(x-t)g(t)\,dt$.

c) Welche der folgenden 2D-Filter sind separierbar?

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

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

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

9 Jacobi, Gauss-Seidel (6 Punkte)

Man betrachte das lineare Gleichungssystem $\mathbf{A}\vec{x} = \vec{b}$ mit $\mathbf{A} = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 2 & 2 & -1 & 0 \\ 0 & 1 & 1 & -1 \\ -1 & 0 & 0 & 2 \end{bmatrix}$, $\vec{b} = \begin{bmatrix} 3 & 0 & 0 & 3 \end{bmatrix}$

Dieses Gleichungssystem soll iterativ gelöst werden, beginnend mit dem Startvektor $\vec{x}^0 = [1, 1, 1, 1]^T$

a) Führen sie einen Schritt des Jacobi-Verfahrens (zum Startwert $\vec{x}^0$) durch.

b) Führen sie einen Schritt des Gauss-Seidel-Verfahrens (zum Startwert $\vec{x}^0$) durch.

c) Konvergiert das JACOBI-Verfahren für die folgenden Matrizen?

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

Begründen Sie Ihre Aussagen stichpunktartig!

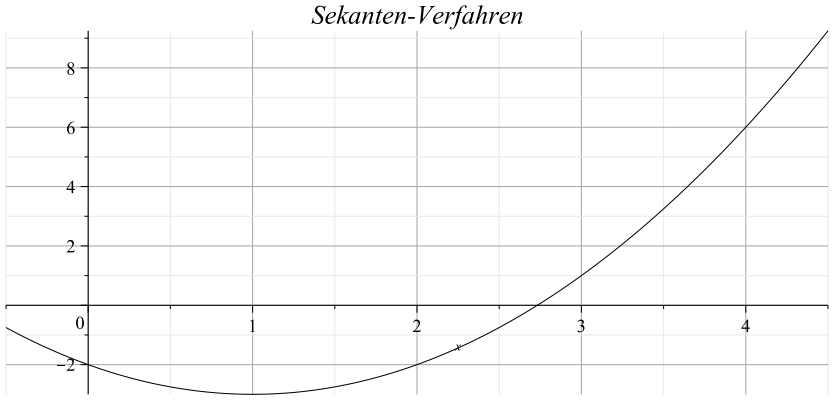
10 Nullstellensuche (11 Punkte)

a) Zunächst soll die Nullstelle iterativ mit dem Sekantenverfahren bestimmt werden.

Startwerte sind: $x_0 = 0$, $x_1 = 4$;

Führen Sie **zeichnerisch** zwei Schritte des Sekantenverfahrens durch. Ermitteln Sie also zeichnerisch x_2 und x_3.

Achten Sie darauf, dass die Zwischenschritte nachvollziehbar sind.

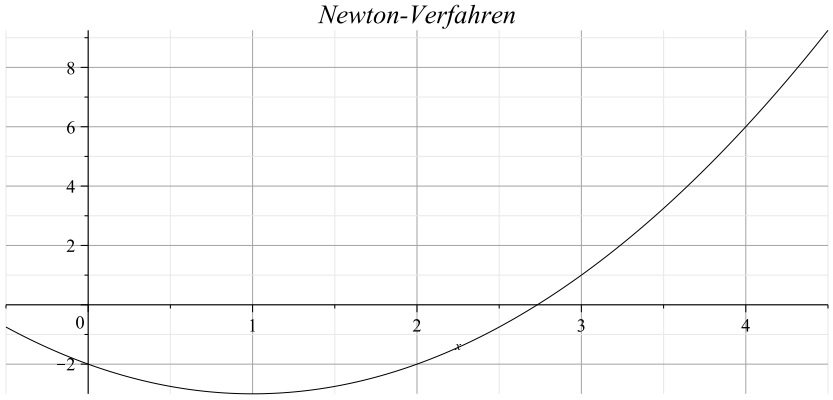


b) Nun soll die Nullstelle iterativ mit dem NEWTON-Verfahren bestimmt werden.

Startwerte ist: $x_0 = 4$.

Führen Sie **zeichnerisch** zwei Schritte des NEWTON-Verfahren durch. Ermitteln Sie also zeichnerisch x_1 und x_2.

Achten Sie darauf, dass die Zwischenschritte nachvollziehbar sind.



c) Gegeben ist das folgende nichtlineare Gleichungssystem:

$$4y + z - y \cdot z - 1 = 0$$
$$4z - y \cdot z - 1 = 0$$

Das Gleichungssystem soll mittels NEWTON-Verfahren gelöst werden. Führen Sie hierzu **eine** NEWTON-Iteration durch. Verwenden Sie als Startwert $\vec{x}^0 = [y^0, z^0]^T = [0, 0]^T$ und berechnen Sie $\vec{x}^1 = [y^1, z^1]^T$.

d) Beantworten Sie folgenden Fragen:

- Konvergiert das NEWTON-Verfahren für beliebige Startwerte?

- Konvergiert das Sekantenverfahren für beliebige Startwerte?

- Konvergenz vorausgesetzt: Was ist die Konvergenzordnung des NEWTON-Verfahrens?