# Algorithmen zur Bestimmung konvexer Hüllen

Frank Reisenhofer

**Diplomarbeit**

Fakultät für Mathematik und Physik
der Universität Bayreuth

September 1995

Betreuer: Prof. Dr. Frank Lempio
Mathematisches Institut
Universität Bayreuth
95440 Bayreuth

This manuscript was prepared with LaTeX.

# Inhaltsverzeichnis

# Notationen

| | |
|---|---|
| $\mathbb{R}^n$ | $n$-dimensionaler Euklidischer Raum |
| $f'$ | Ableitung von $f$ |
| $< x, y >$ | Euklidisches Skalarprodukt der Vektoren $x, y \in \mathbb{R}^n$ |
| $[x, y]^*$ | Verbindungsgerade zwischen den Punkten $x, y \in \mathbb{R}^n$ |
| $B_r(\tilde{x})$ | abgeschlossene Kugel mit Radius $r$ um $\tilde{x}$ |
| $C[a, b]$ | Raum der stetigen Funktionen auf $[a, b]$ |
| $L^\infty[a, b]$ | Raum der wesentlich beschränkten Funktionen auf $[a, b]$ |
| $L^2[a, b]$ | Raum der quadrat-integrierbaren Funktionen auf $[a, b]$ |
| $AC[a, b]$ | Raum der absolut stetigen Funktionen auf $[a, b]$ |
| $E_n$ | n-dimensionale Einheitsmatrix |
| N | Diskretisierungsparameter / Anzahl der Teilintervalle |
| a,b | Intervallgrenzen |
| h | Schrittweite |
| $f(\cdot)$ | reelle Funktion |
| $F(\cdot)$ | Mengenwertige Abbildung |
| $dim\ A$ | Dimension der Menge $A$ |
| $\bigvee$ | entspricht dem üblichen $\forall$ |
| $\bigwedge$ | entspricht dem gewöhnlichen $\exists$ |
| VATER(v) | Vaterknoten eines Knotens $v$ |
| BRUDER(v) | Bruderknoten eines Knotens $v$ |

## Kapitel 0

# Mathematische Grundlagen

Diese Arbeit beschäftigt sich mit Algorithmen zur Bestimmung konvexer Hüllen. In einem einführenden Kapitel werden grundlegende Begriffe, wie konvexe Hülle und affine Hülle definiert. Anschließend wird darauf eingegangen, welche Operationen mit Mengen durchgeführt werden können. Danach wird dargelegt, daß die Bestimmung von konvexen Hüllen bei einem direkten Ansatz zur Approximation von mengenwertigen Integralen von entscheidender Bedeutung ist. Die Bestimmung der konvexen Hülle einer endlichen Anzahl von Punkten mit Hilfe geeigneter Programme findet zahlreiche weitere Anwendungen, etwa bei der Mustererkennung [1] oder der Bildverarbeitung [15]. Auf diese beiden Anwendungen wird in dieser Arbeit jedoch nicht näher eingegangen.

Nach dieser Einführung werden Algorithmen zur Bestimmung der konvexen Hülle einer endlichen Anzahl von Punkten aus der Ebene ausführlich behandelt. Dabei werden Ansätze vorgestellt, hinter denen zwar eine "einfache" Idee steckt, die jedoch eine recht lange Laufzeit besitzen. Nachdem eine Aussage über die optimale Laufzeit eines Algorithmus zur Lösung des Problems in der Ebene bewiesen worden ist, werden Algorithmen beschrieben, die dieses Laufzeitverhalten besitzen. Darüber hinaus wird ein Algorithmus erläutert, der die konvexe Hülle einer endlichen Punktemenge online bestimmt, sowie ein Ansatz bei dem diese Hülle lediglich approximiert wird.

Anschließend wird auf die Implementierung eines Algorithmus eingegangen, der aufgrund seines optimalen Laufzeitverhaltens realisiert wurde. Dieser Algorithmus wird im folgenden getestet, indem mit Hilfe eines direkten Ansatzes mengenwertige Integrale im Aumannschen Sinne approximiert werden. Ein wichtiges Resultat der numerischen Ergebnisse ist, daß wir bei einer großen Klasse von Aumannintegralen einen Zusammenhang zwischen dem gewählten Diskretisierungsparameter und der Anzahl der Eckpunkte der approximierenden Menge beobachten können. Dieser Zusammenhang wird von mir theoretisch begründet. Da es dieses Resultat ermöglicht, eine obere Schranke für die Anzahl

der Eckpunkte einer approximierenden Menge anzugeben, ist es möglich geworden das Laufzeitverhalten der benutzten direkten Methoden zur Approximation von mengenwertigen Integralen abzuschätzen.

Um das Gebiet der Algorithmen zur Bestimmung konvexer Hüllen abzurunden, werden im letzten Kapitel Methoden vorgestellt, die die konvexe Hülle einer endlichen Punktemenge aus dem $\mathbb{R}^n$ ermitteln können.

Das Konzept, das der Definition der konvexen Hülle einer Menge von Punkten zugrundeliegt, ist leicht zu verstehen:

## 0.1 Grundlegende Definitionen

**Definition 0.1.1:**
Sei $M$ eine Teilmenge des $\mathbb{R}^n$ .
$M$ ist *konvex*, falls für alle $x \in M$ und für alle $y \in M$ sowie für alle $c \in ]0,1[$ gilt:

$$(1-c) \cdot x + c \cdot y \in M$$

Der Durchschnitt aller konvexen Mengen, die eine Teilmenge $S$ des $\mathbb{R}^n$ enthalten, wird als *konvexe Hülle* $co(S)$ bezeichnet. Also ist

$$co(S) = \bigcap_{B \cap S = S} \{B : Bkonvex\}.$$

◇

Auf ähnliche Weise definieren wir die affinen Mengen bzw. die affinen Hüllen:

**Definition 0.1.2:**
Sei $M$ eine Teilmenge des $\mathbb{R}^n$.
$M$ ist *affin*, falls für alle $x \in M$, für alle $y \in M$ und für alle $c \in \mathbb{R}^n$ gilt:

$$(1-c) \cdot x + c \cdot y \in M$$

Analog zu oben wird der Durchschnitt aller affinen Mengen, die eine Teilmenge $S$ des $\mathbb{R}^n$ enthalten, als *affine Hülle* bezeichnet. Damit ist

$$aff(S) = \bigcap_{B \cap S = S} \{B : Baffin\}.$$

◇

**Beispiel 0.1.3:** Gegeben sei eine Menge $S = \{x, y\}$ mit $x, y \in \mathbb{R}^2 (x \neq y)$.
Dann ergibt sich als konvexe Hülle von $S$ die Verbindungsgerade zwischen den beiden Punkten $x$ und $y$.
Die Gerade, die durch beide Punkte verläuft, stellt die affine Hülle der Menge $S$ dar. ◇

Damit wir noch eine bessere Vorstellung von konvexen Hüllen erhalten, benötigen wir eine weitere Definition:

**Definition 0.1.4:**
Eine *polyedrische Menge* ist der Durchschnitt einer endlichen Anzahl von Halbräumen. ◇

Nach ([11], S. 43-47) gilt nun folgender Zusammenhang zwischen konvexen Hüllen und polyedrischen Mengen:

**Satz 0.1.5:** *Die konvexe Hülle einer endlichen Anzahl von Punkten ist eine beschränkte polyedrische Menge. Die Umkehrung gilt ebenso.* ◇

## 0.2 Mengenoperationen in linearen Räumen

Für die Anwendung ist es sinnvoll auf Mengen Operationen zu definieren, da wir mit Mengen - wenn möglich - genauso wie mit reellen Zahlen rechnen möchten.

**Definition 0.2.1:**
Sei $X$ ein linearer Raum über $\mathbb{R}$ (i.a. auch über $\mathbb{C}$ möglich!).
Die *Addition* zweier Mengen $A, B \subset X$ wird wie folgt definiert:

$$A + B := \{x \in X | \exists a \in A; \exists b \in B : a + b = x\}.$$

Die *skalare Multiplikation* ist gegeben durch:

$$\lambda \cdot A := \{x \in X | \exists a \in A : \lambda \cdot a = x\}.$$

Die Multiplikation einer Matrix $M$ mit einer Menge wird definiert durch:

$$M \circ A := \{M \cdot a | a \in A\}\ M \in \mathbb{R}^{m \times n}, A \subset \mathbb{R}^n.$$

◇

## 0.3 Mengenwertige Integrale

Es stellt sich nun die Frage, wo solche Mengenoperationen benutzt werden können. Einer Anwendung liegt die folgende Überlegung zu Grunde: Bei Integralen im Riemannschen Sinne verwenden wir Quadraturformeln (in der Regel Newton-Cotes-Formeln), um uns eine numerische Näherungslösung zu beschaffen. Es liegt nun nahe, bei Betrachtung von mengenwertigen Integralen auf analoge Weise eine Näherungslösung zu bestimmen. Bevor wir darauf eingehen können, erweitern wir den Integralbegriff:

**Definition 0.3.1:**
Seien $a, b \in \mathbb{R}$. Sei $I = [a, b]$ ein Intervall.
Sei $F : I \to \wp(\mathbb{R}^n)$ eine mengenwertige Abbildung. So bezeichnet man

$$\int_I F(t)dt := \{\int_I f(t)dt | f(t) \in F(t), \forall t \in I, f(\cdot) \in L_1(I)^n\}$$

als *Aumann-Integral* von $F(\cdot)$. $\diamond$

Aumannintegrale müssen bei optimalen Steuerungsproblemen berechnet werden, wie beim Mayerschen Kontrollproblem :

$$\min x_1(b)$$

unter den Nebenbedingungen:

$$\left.\begin{array}{l} x(\cdot) \in AC(I)^n \\ x'(t) \in A(t) \cdot x(t) + B(t) \cdot U \ \forall' t \in I := [a, b] \\ x(a) \in Y_0 \end{array}\right\} (P1)$$

wobei

$A(\cdot)$ eine integrierbare n $\times$ n-Matrixfunktion,
$B(\cdot)$ eine integrierbare n $\times$ m Matrixfunktion,
$U \subset \mathbb{R}^m$ eine nichtleere, kompakte Steuerungsmenge und
$Y_0 \subset \mathbb{R}^n$ eine kompakte, konvexe, nichtleere Anfangsmenge ist.

Um das Problem lösen zu können, muß man zunächst die sogenannte erreichbare Menge bestimmen.

**Definition 0.3.2:**
Die Menge

$$\Re(a, b, y_0) := \{x \in \mathbb{R}^n | \exists \, Lösung \, x(\cdot) \, für \, (P1) \, mit \, x(a) = y_0; x(b) = x\}$$

bezeichnet die *erreichbare Menge* für den Startzeitpunkt $a$, den Startwert $y_0$ zum Endzeitpunkt $b$. $\diamond$

Die erreichbare Menge obigen Problems läßt sich durch ein mengenwertiges Integral berechnen. Es gilt (vgl. [2]):

$$\Re(a, b, y_0) = \phi(b, a) \cdot Y_0 + \int_a^b \phi(b, \tau) \cdot B(\tau) \cdot U d\tau$$

Dabei bezeichnet $\phi(t, \tau)$ das Fundamentalsystem der homogenen Ausgangsgleichung mit $\phi(\tau, \tau) = E_n$.
Also gilt: $\frac{d}{dt}\phi(t, \tau) = A(t) \cdot \phi(t, \tau); \phi(\tau, \tau) = E_n$.

Kennt man die erreichbare Menge $\Re(a,b,Y_0)$, so kann man mit gewöhnlichen Optimierungsalgorithmen eine Lösung des Minimierungsproblems erhalten.

Es stellt sich uns nun die Frage, wie wir das Integral

$$\int_a^b \phi(b,\tau)\cdot B(\tau)\cdot U d\tau =: \int_a^b F(\tau)d\tau$$

näherungsweise berechnen können. Bei gewöhnlichen Integralen $\int_a^b f(t)dt$ wendet man z. B. Newton-Cotes-Formeln wiederholt an, d.h. die Formeln werden nicht auf das gesamte Intervall $[a,b]$ angewandt, sondern auf kleinere äquidistante Intervalle, und anschließend werden die Näherungswerte für die Teilintervalle addiert (siehe [18], S. 114-118).

Beispiele für iterierte Anwendung von Newton-Cotes-Formeln sind der nachfolgenden Tabelle zu entnehmen:

| **Regel** | **Summenformel** |
|---|---|
| Treppenregel | $h\cdot[f(a)+f(a+h)+\ldots+f(b-h)]$ |
| Trapezregel | $h\cdot[\frac{f(a)}{2}+f(a+h)+\ldots+\frac{f(b)}{2}]$ |
| Simpsonregel | $\frac{h}{3}[f(a)+4f(a+h)+2f(a+2h)+\ldots+4f(b-h)+f(b)$ |

Dieses Vorgehen können wir auf den mengenwertigen Fall übertragen. Als Ansatz ergibt sich hierfür:

$$\int_a^b F(\tau)d\tau \approx \frac{h}{2}\left(F(a)+2\sum_{i=1}^{N-1}F(t_i)+F(b)\right)\ (t_i=a+i\cdot h,\ i=1,\ldots,N) \quad (0.1)$$

als *Trapezregel für den mengenwertigen Fall*, bzw.

$$\int_a^b F(\tau)d\tau \approx \frac{h}{3}\left(F(a)+4\sum_{i=1}^{\frac{N}{2}}F(t_{2i-1})+2\sum_{i=1}^{\frac{N-2}{2}}F(t_{2i})+F(b)\right) \quad (0.2)$$

als *Simpsonregel für den mengenwertigen Fall*. Bei letzterer muß der Diskretisierungsparameter $N$ geradzahlig sein. Die bei der Approximation auftretenden Fehler sollen an dieser Stelle nicht berücksichtigt werden.

Betrachten wir den Fall, daß die mengenwertige Abbildung $F(\cdot)$ lauter nichtleere, konvexe und kompakte Bilder besitzt, so gilt zunächst:

$$\int_a^b F(t)dt = \int_a^b co\,(F(t))\,dt =: \int_a^b \tilde{F}(t)dt$$

Durch Anwendung von (0.1) ergibt sich:

$$\begin{aligned}\int_a^b F(t)dt &\approx \frac{h}{2}\left(\tilde{F}(a) + 2\sum_{i=1}^{N-1}\tilde{F}(t_i) + \tilde{F}(b)\right) \\ &= \frac{h}{2}\left(co\left(F(a)\right) + 2\sum_{i=1}^{N-1} co\left(F(t_i)\right) + co\left(F(b)\right)\right) \qquad (0.3)\end{aligned}$$

und analog mit (0.2)

$$\begin{aligned}\int_a^b F(t)dt \approx \frac{h}{3}\Bigg(&co(F(a)) + 4\sum_{i=1}^{\frac{N}{2}} co(F(t_{2i-1})) \\ &+2\sum_{i=1}^{\frac{N-2}{2}} co(F(t_{2i})) + co(F(b))\Bigg) \qquad (0.4)\end{aligned}$$

Somit können wir Aumannintegrale approximieren, indem wir eine Menge von skalierten konvexen Mengen addieren.
Verfügen wir über Algorithmen, die die konvexe Hülle einer endlichen Punktmenge ermitteln können, so müssen wir lediglich noch geeignete Funktionen implementieren, mit deren Hilfe wir Mengen addieren bzw. skalieren können. Algorithmen, welche die konvexe Hülle einer endlichen Zahl von Punkten ermitteln, stehen im Mittelpunkt weiterer Betrachtungen.

# Kapitel 1

# Algorithmen zur Bestimmung konvexer Hüllen in der Ebene

Zur Lösung numerischer Probleme oder ganz allgemein bei Problemen aus der Informatik, gibt es in der Regel mehrere Algorithmen, mit denen wir ein gegebenes Problem lösen können. Die Algorithmen möchten wir miteinander vergleichen. Als mögliche Vergleichskriterien bieten sich uns an:

1. Benötigter Speicherplatz eines Programmes sowie dessen gesamte Speicherplatzanforderung.
2. Die Genauigkeit der gefundenen Lösung in Bezug auf eine exakte Lösung.
3. Die Konvergenzgeschwindigkeit / Konvergenzordnung.
4. Die benötigte Laufzeit.

(1) spielt in einer Zeit, da wir mit GigaByte-Festplatten arbeiten, wohl eher eine untergeordnete Rolle. Die Kriterien (2) und (3) werden wir bei der Anwendung von Algorithmen, die die konvexe Hülle einer endlichen Punktmenge bestimmen, in der Numerik genauer betrachten. Die im folgenden vorgestellten Algorithmen werden wir hinsichtlich ihrer Laufzeit miteinander vergleichen. Dazu benötigen wir:

**Definition 1.0.1:**
Sei $t$ die Laufzeit eines Algorithmus und $f : \mathbb{N} \to \mathbb{R}$ eine reelle Funktion. So bedeutet ein Ausdruck der Form $O\left(f(\tilde{n})\right)\,(\tilde{n} \in \mathbb{N})$, daß es eine Zahl $M > 0$ und ein $n_0 \in \mathbb{N}$ mit $t \leq M|f(n)|$ für alle $n \geq n_0$ gibt. Ein Ausdruck der Form $\Omega\left(f(\tilde{n})\right)\,(\tilde{n} \in \mathbb{N})$ bedeutet, daß es eine Zahl $c > 0$ und ein $n_0 \in \mathbb{N}$ gibt, so daß $t \geq c|f(n)|$ für alle $n \geq n_0$. $\diamond$

**Bemerkung:** $\Omega(f(N))$ ist eine untere Schranke für die Laufzeit eines Programms. $\diamond$

In der Ebene, also im zweidimensionalen Raum, stellt sich uns folgendes Problem:
Gegeben sei eine Menge $M = \{p_1, \ldots, p_N\}$ $(p_i \in \mathbb{R}^2;\ i = 1, \ldots, N;\ N \in \mathbb{N})$ endlich vieler Punkte aus der Ebene.
Gesucht sind alle die Punkte $\{p_{i_0}, \ldots, p_{i_j}\} \subset M$ $(j \in \{0, \ldots, N-1\})$, welche Eckpunkte der zugehörigen konvexen Hülle $co(M)$ sind.

**Definition 1.0.2:**
Sei $M$ eine Menge endlich vieler Punkte.
Dann bezeichnen wir mit $EP(M)$ alle Punkte von $M$, die genau die Eckpunkte der zugehörigen konvexen Hülle darstellen.
Mit $SEP(M)$ benennen wir die sortierte $EP(M)$ Menge, d.h. die Punkte aus $EP(M)$ sind so angeordnet, daß das Polytop, das die konvexe Hülle darstellt, durch Verbinden aufeinanderfolgender Punkte gezeichnet werden kann. $\diamond$

**Beispiel 1.0.3:** Gegeben seien die Punkte

$$p_1 = (1,0), p_2 = (0,1), p_3 = (0,0), p_4 = (0.5, 0.5) \text{ und } p_5 = (1,1).$$

In diesem Fall erhalten wir:

$$EP(\{p_1, p_2, p_3, p_4, p_5\}) = \{p_1, p_2, p_3, p_5\}$$
$$SEP(\{p_1, p_2, p_3, p_4, p_5\}) = \{p_2, p_3, p_1, p_5\}$$

$\diamond$

## 1.1 Zwei einfache Ansätze

Möchten wir die Menge $SEP(M)$ bestimmen, so gibt es hierfür zwei mögliche Konzepte. Bei dem einen überlegen wir uns Kriterien dafür, wann ein Punkt $p \in M$ ein Eckpunkt der konvexen Hülle ist. Methoden, denen solche Entscheidungsmerkmale zugrunde liegen, bezeichnen wir als direkte Methoden (DM). Im Gegensatz dazu gibt es die indirekten Methoden (IM). Bei diesen suchen wir nach Kriterien, wann ein Punkt $p \in M$ nicht zu $SEP(M)$ gehört. Ein solches Kriterium ist das folgende:

**Kriterium 1 (K1):**
Finden wir drei Punkte $p_1, p_2, p_3 \in M$, die ein Dreieck $\triangle\, p_1\, p_2\, p_3$ derart festlegt, daß ein Punkt $p \in M$ im Inneren dieses Dreiecks liegt, so ist $p \in M$ kein Eckpunkt der konvexen Hülle $co(M)$.

Ein Algorithmus (A1), der die Menge $SEP(M)$ mit Hilfe von (K1) ohne Berücksichtigung von Spezialfällen (z.B. $|M| < 3$) bestimmt, sieht folgendermaßen aus:

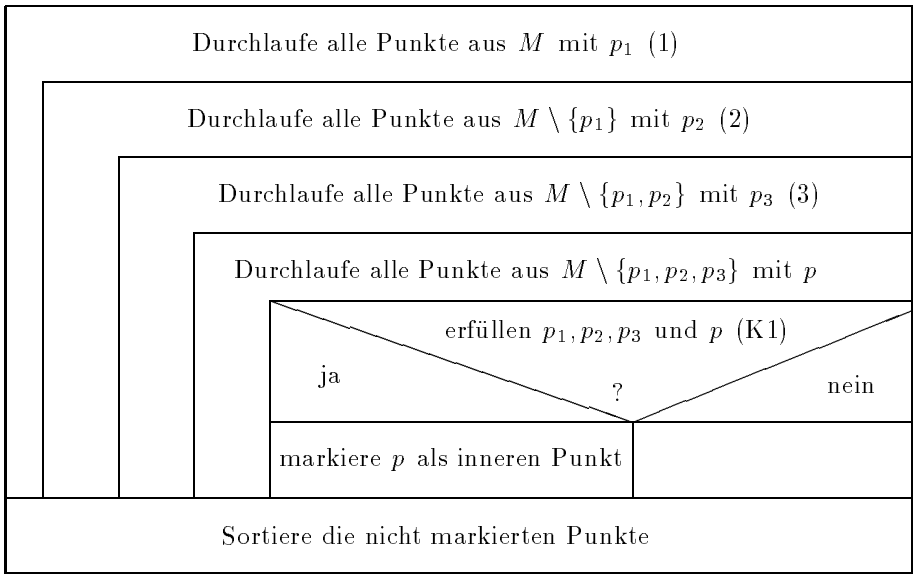



Abbildung 1.1: Struktogramm für (A1)

Für das Laufzeitverhalten von (A1) gilt der Satz:

**Satz 1.1.1:** *Verwendet man den Algorithmus A1 zur Bestimmung der Menge SEP(M), so benötigt man einen Zeitaufwand von* $O(N^4)$.

*Beweis.* Im Algorithmus A1 müssen vier ineinandergeschachtelte Schleifen durchlaufen werden, wofür jeweils ein Aufwand von $O(N)$ anfällt. Zur Abarbeitung aller Schleifen benötigen wir folglich einen Aufwand von $O(N^4)$. Das Kriterium (K1) kann man, unabhängig von $N$, in konstanter Zeit lösen (z. B. durch die Bestimmung der Vorzeichen von drei Determinanten, siehe 2.2.1). Zum Sortieren der Punkte braucht man $O(N \log N)$ Zeit. Insgesamt bleibt somit ein Aufwand von $O(N^4)$. □

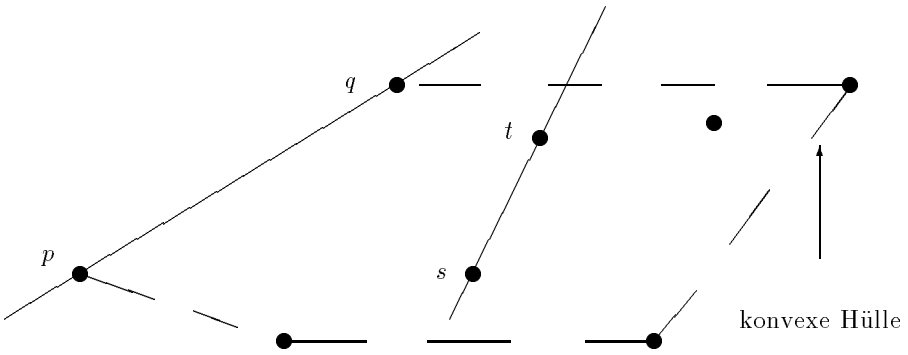



Abbildung 1.2: Veranschaulichung von 1.1.2

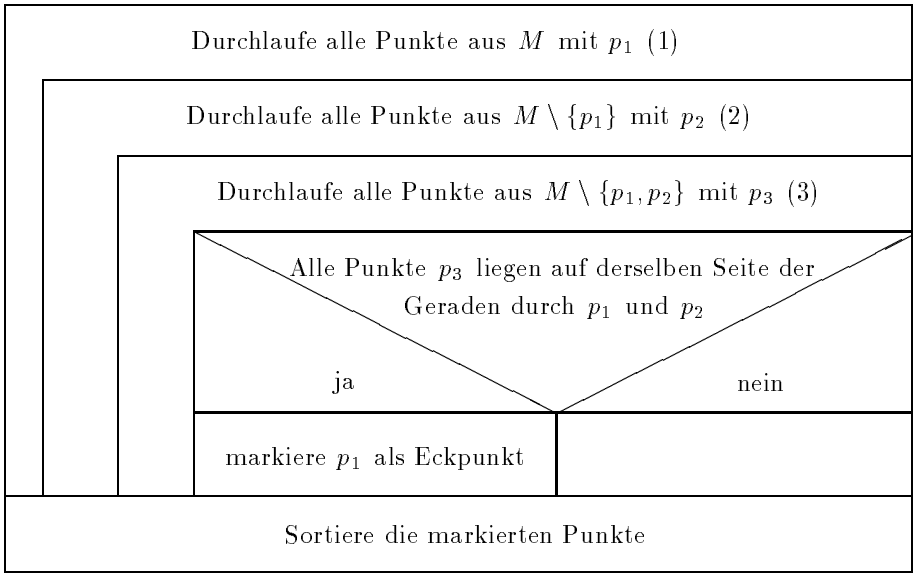
Um die konvexe Hülle einer endlichen Punktemenge direkt bestimmen zu können, benötigen wir Kriterien, die eine Aussage darüber liefern, wann ein beliebiger Punkt aus einer Punktmenge einen Eckpunkt der konvexen Hülle darstellt. Ein solches resultiert aus:

**Satz 1.1.2:** *Die Verbindungsgerade $l$, die durch zwei Punkte $p_1, p_2$ einer Punktmenge definiert wird, ist eine Kante der konvexen Hülle, genau dann wenn alle übrigen Punkte auf $l$ oder nur auf einer Seite der Geraden liegen ([19], Theorem 2.4.7).*

Den Inhalt des Satzes veranschaulicht uns Abbildung 1.2. In dieser ist die Strecke $\overline{pq}$ eine Kante der konvexen Hülle, während die Strecke $\overline{st}$ keine ist. Wir erhalten als Entscheidungskriterium:

**Kriterium 2(K2):**
Ein Punkt $p$ aus einer endlichen Punktmenge $M$ ist genau dann ein Eckpunkt der konvexen Hülle, falls es einen weiteren Punkt $q \in M$ gibt, so daß $p$ und $q$ die Voraussetzungen aus 1.1.2 erfüllen.

Ein Algorithmus (A2) der mittels (K2) die Menge $SEP(M)$ ermittelt, sieht unter Vernachlässigung von Spezialfällen wie folgt aus:




Der Zeitaufwand ist von den drei ineinandergeschachtelten Schleifen (1), (2) und (3) abhängig. Es gilt demzufolge der Satz:

**Satz 1.1.3:** *Der Algorithmus (A2) benötigt zur Bestimmung von* $SEP(M)$ *einen Aufwand von* $O(N^3)$. ◇

A2 löst unser Problem schneller als A1. Uns stellt sich somit die Frage, ob wir für die Laufzeit eines Algorithmus, der die Menge $SEP(M)$ erzeugt, eine untere Schranke - also $\Omega(N)$- angeben können. Diese gibt der nächste Satz an:

**Satz 1.1.4:** *Zur Bestimmung der Menge* $SEP(M)$ *benötigen wir in der Ebene* $\Omega(N \log N)$ *Zeit.* ◇

*Beweis.* Die Grundlage des Beweises bildet folgendes Lemma:

**Lemma 1.1.5:** *Benötigen wir zum Lösen eines Problems A einen mittleren Zeitaufwand von* $T(N)$, *und ist dieses Problem* $\tau(N)$*-transformierbar auf ein Problem B, d.h. man braucht für die Transformation der Eingabe für das Problem A auf eine geeignete Eingabe für das Problem B sowie für die Abbildung der Ausgabe des Problems B auf eine richtige Lösung des Problems A insgesamt* $O(\tau(N))$ *Zeit, so benötigen wir zum Lösen des Problems B einen Zeitaufwand von mindestens* $T(N) - O(\tau(N))$. ◇

*Beweis.* Hierzu führen wir einen Widerspruchsbeweis. Angenommen es gäbe eine kleinere untere Schranke für den Aufwand zur Lösung des Problems $B$. In

diesem Fall könnten wir den mittleren Zeitaufwand zur Lösung des Problems $A$ reduzieren, indem wir die Eingabe für das Problem $A$ auf eine Eingabe für das Problem $B$ transformieren. Anschließend lösen wir dieses und machen die Transformation rückgängig. Damit wäre der mittlere Zeitaufwand verkleinert worden. Dies ist ein Widerspruch. □

Betrachte nun die Probleme:

$B$:={ Gegeben sei eine Menge $M \subset \mathbb{R}^2$ mit $|M| = N$. Bestimme $SEP(M)$. }
$A$:={ Gegeben seien N reelle Zahlen. Gesucht ist eine Sortierung der Zahlen. }

Zur Lösung des Sortierproblems $A$ benötigen wir einen mittleren Aufwand von $T(N) = O(N \log N)$ [10]. Diesen Aufwand besitzen nach [10] der Quick-, Heap- und der Mischsortieralgorithmus.
Wir zeigen nun, daß das Problem $A$ $N$-transformierbar auf $B$ ist. Betrachte dazu die Transformation:

$$T_1 : \mathbb{R} \to \mathbb{R}^2,\ x \mapsto (x, x^2)$$

Zur Transformierung von $N$ reellen Zahlen mit $T_1$ ist ein Aufwand von $O(N)$ erforderlich, da jede Zahl genau einmal zwecks der Abbildung durchlaufen werden muß. Zur Rücktransformation benutzen wir

$$T_2 : \mathbb{R}^2 \to \mathbb{R},\ (x, y) \mapsto x$$

wofür ein Aufwand von $O(N)$ anfällt. Das Problem $A$ kann nun wie folgt gelöst werden:

1. Transformiere die in einem Feld gespeicherten Zahlen mit $T_1$.
2. Bestimme $SEP(M)$ und speichere die Punkte in einer verketteten Liste.
3. Transformiere $SEP(M)$ vermöge $T_2$ in ein Array.

Damit wir auf diese Weise sortieren können, muß der Algorithmus, der $SEP(M)$ bestimmt, so ausgelegt sein, daß der am weitesten außen links stehende Punkt als erster in der Menge hinterlegt wird. Ist dies nicht der Fall, so muß vor der Transformation (3) der am weitesten außen links stehende Punkt $\tilde{p}$ in der einfach verketteten Liste ermittelt werden. Hierfür fällt ein Aufwand von $O(N)$ an. Anschließend werden die Punkte beginnend bei diesem Punkt $\tilde{p}$ transformiert. Ist das Ende der Liste erreicht, so fährt man mit dem ersten Listenelement fort, bis wieder $\tilde{p}$ erreicht ist. Nachdem (1) durchgeführt worden ist, liegen alle Punkte in der Ebene auf einer Parabel. Somit ist jeder Punkt ein Eckpunkt und ist in $SEP(M)$ enthalten. Es geht folglich kein Punkt bei der Transformation verloren. □

## 1.2 Der Algorithmus von GRAHAM

Unser angestrebtes Ziel ist es nun einen Algorithmus zu entwickeln, der die Menge $SEP(M)$ mit einem Aufwand von $O(N \log N)$ bestimmt.

Bei der Entwicklung eines Solchen ist von entscheidender Bedeutung, daß wir in einem ersten Schritt die Punkte aus der Menge $M$ sortieren und erst in einem Zweiten mit Hilfe eines neuen Kriteriums innere Punkte eliminieren. Bei den Algorithmen A1 und A2 haben wir zunächst innere Punkte bestimmt, und haben erst später die verbleibende Menge von Eckpunkten sortiert.

Um die Punktmenge zu sortieren, gehen wir folgendermaßen vor. Zunächst wird ein innerer Punkt ermittelt. Hierzu können wir Kriterium (K1) verwenden. Beispielsweise bestimmen wir drei Punkte $p_1, p_2, p_3 \in M$, die ein Dreieck festlegen. Der Schwerpunkt $S$ des Dreiecks ist nach (K1) ein innerer Punkt. Nun werden alle Punkte aus $M$ so transformiert, daß der Punkt $S$ den Ursprung des neuen Koordinatensystems bildet. Die Punkte $p_1, \ldots, p_N$ werden anschließend aufsteigend nach dem Winkel sortiert, den die positive x-Achse des neuen Koordinatensystems mit der Strecke $\overline{0p_i}$ einschließt. Falls für mehrere Punkte $p_{i_1}, \ldots, p_{i_k}$ $(k \leq N;\ \{i_1, \ldots, i_k\} \subset \{1, \ldots, N\})$ die gleichen Winkel errechnet werden, so sortieren wir diese aufsteigend nach ihrer Entfernung zum Ursprung. Auf diese Weise erhalten wir die sortierte Punktemenge $\tilde{M}$.

**Beispiel 1.2.1:** Gegeben seien die Punkte

$$p_1 = (-1,-1), p_2 = (-1,3), p_3 = (0,0), p_4 = (1,2) \; und \; p_5 = (2,0).$$

$p_1, p_4, p_5$ bestimmen ein Dreieck mit $S = (1,1)$. Nach der Transformation erhalten wir folgende Punkte:

$$\tilde{p_1} = (-2,-2), \tilde{p_2} = (-2,2), \tilde{p_3} = (-1,-1), \tilde{p_4} = (0,1) \; und \; \tilde{p_5} = (1,-1).$$

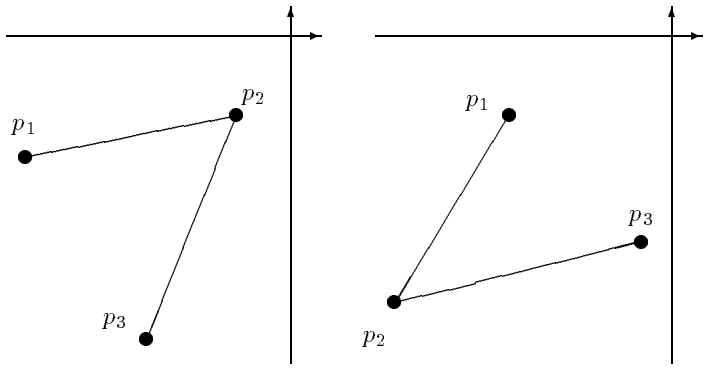
Bezeichnet man mit $\phi_1, \ldots, \phi_5$ die zugehörigen wie oben angegebenen Winkel, so stehen die Winkel in folgender Beziehung:

$$\phi_4 < \phi_2 < \phi_1 = \phi_3 < \phi_5$$

Da $\phi_1 = \phi_3$ müssen die Punkte $\tilde{p_1}$ und $\tilde{p_3}$ nach ihrer Entfernung zum Ursprung sortiert werden. Da $\tilde{p_3}$ näher an diesem liegt, muß der Punkt $\tilde{p_3}$ bei der Sortierung vor $\tilde{p_1}$ liegen. Als sortierte Punktmenge $\tilde{M}$ ergibt sich somit:

$$\tilde{M} = \{\tilde{p_4}, \tilde{p_2}, \tilde{p_3}, \tilde{p_1}, \tilde{p_5}\}$$

$\diamond$




$p_1, p_2, p_3$ sind rechtsorientiert  $p_1, p_2, p_3$ sind linksorientiert

Abbildung 1.3: Veranschaulichung von linksorientiert und rechtorientiert

Nachdem die Punkte auf diese Weise sortiert und transformiert worden sind, überlegen wir uns ein neues Kriterium dafür, wann ein Punkt $p \in M$ ein innerer Punkt ist. Dazu definieren wir:

**Definition 1.2.2:**
Gegeben seien $p_1, p_2, p_3 \in M \subset \mathbb{R}^2$. Falls der gegen den Uhrzeigersinn orientierte Winkel zwischen den Vektoren $p_2 - p_1$ und $p_3 - p_2$ größer als $\pi$ ist, so sind die Punkte $p_1, p_2$ und $p_3$ *rechtsorientiert*. Ansonsten sind die Punkte *linksorientiert* angeordnet. ◇

Diese Definition veranschaulicht uns Abbildung 1.3.

**Bemerkung:** Würde ein Fußgänger von einem Punkt $p_1$ zu einem Punkt $p_3$ laufen und dabei zwischenzeitlich den Punkt $p_2$ passieren, so müßte der Fußgänger, falls die Punkte $p_1, p_2$ und $p_3$ rechtsorientiert liegen, am Ort $p_2$ nach rechts abbiegen, während er sich in der anderen Situation nach links orientieren müßte. ◇

Es bietet sich nun folgendes Entscheidungskriterium an:

**Kriterium 3(K3):**
Betrachten wir drei aufeinanderfolgende Punkte $p_i, p_{i+1}, p_{i+2}$ aus $\tilde{M}$, so ist $p_{i+1}$ kein Eckpunkt, wenn die Punkte $p_i, p_{i+1}, p_{i+2}$ rechtsorientiert sind.

In diesem Fall liegt $p_{i+1}$ im Inneren des Dreiecks $\triangle p_i 0 p_{i+2}$. Nach Kriterium K1 ist $p_{i+1}$ ein innerer Punkt. Für diese Argumentation ist zu beachten, daß der Ursprung ein innerer Punkt von $co(\tilde{M})$ ist.
Wir können die Menge $SEP(M)$ bestimmen, indem wir die Menge $SEP(\tilde{M})$ errechnen, und anschließend die Transformation rückgängig machen.
Um alle inneren Punkte ermitteln zu können, gehen wir folgendermaßen vor: Wir betrachten drei aufeinanderfolgende Punkte $p_i, p_{i+1}, p_{i+2}$ aus $\tilde{M}$, die noch nicht als innere Punkte erkannt wurden, und überprüfen, ob sie rechtsorientiert liegen. Ist dies der Fall, so ist $p_{i+1}$ ein innerer Punkt. Wir markieren $p_{i+1}$ als inneren Punkt, und eliminieren ihn somit aus der Liste der zu untersuchenden Punkte. Als nächstes betrachten wir die Punkte $p_{i-1}, p_i, p_{i+2}$. Andernfalls untersuchen wir die Punkte $p_{i+1}, p_{i+2}, p_{i+3}$. Nachdem alle denkbaren Tripel von aufeinanderfolgenden Punkten untersucht worden sind, bleibt die Menge $SEP(\tilde{M})$ übrig. Zusammenfassend ergibt sich als Ablaufschema:

1. Bestimme einen inneren Punkte $\tilde{p}$.
2. Führe eine Transformation der gegebenen Punktemenge $M$ auf $\tilde{M}$ so durch, daß $\tilde{p}$ den Ursprung eines neuen Koordinatensystems festlegt.
3. Sortiere die Punktmenge $\tilde{M}$ aufsteigend nach dem Sortierkriterium "polarer Winkel" und "Entfernung zum Ursprung".
4. Bestimme mittels (K3) die inneren Punkte von $\tilde{M}$.
5. Transformiere $SEP(\tilde{M})$ auf $SEP(M)$ durch Umkehrung der Transformation aus. 2

**Bemerkung:** Da bei diesem Ansatz, der von Graham vorgestellt wurde, die Punkte quasi gegen den Uhrzeigersinn abgetastet werden, bezeichnen wir diesen Algorithmus als *Grahamsche Abtastung.* ◇

Wir interessieren uns für die Laufzeit des Algorithmus von GRAHAM. Hierüber gibt nachfolgender Satz Auskunft:

**Satz 1.2.3:** *Zur Bestimmung der Menge $SEP(M)$ mittels der Grahamschen Abtastung benötigen wir einen Aufwand von $O(N \log N)$, wobei $N$ der Anzahl der Punkte in $M$ entspricht.* ◇

*Beweis.* Um den Gesamtaufwand abschätzen zu können, müssen wir den Aufwand ermitteln, der für die Schritte (1)-(5) anfällt. Der Gesamtaufwand ergibt sich als Summe der Aufwände für die einzelnen Schritte.
Zur Bestimmung von $\tilde{p}$ in (1) müssen wir drei Punkte aus $M$ finden, die ein Dreieck festlegen. Da mit konstantem Zeitaufwand überprüft werden kann, ob

drei Punkte ein Dreieck bestimmen, und im schlimmsten Fall alle Punkte betrachten werden müssen, bis wir drei Punkte gefunden haben, die ein Dreieck festlegen, ist für diesen Schritt ein Zeitaufwand von $O(N)$ erforderlich.
Für die Transformationsschritte (2) und (5) ist ebenfalls jeweils ein Aufwand von $O(N)$ erforderlich, da jeder Punkt aus $M$ transformiert werden muß, und bei der Transformation eines Punktes lediglich Additions- bzw. Subtraktionsoperationen durchgeführt werden müssen.
Aus der Informatik (vgl. [10]) wissen wir, daß zur Lösung des Sortierproblems (3) ein Aufwand von $O(N \log N)$ zu veranschlagen ist.
Bei Schritt (4) schlägt ein Aufwand von $O(N)$ zu Buche, da bei der Überprüfung der Orientierung von drei Punkten ein konstanter Zeitaufwand anfällt. Nach jeder Überprüfung müssen wir entweder die nächsten drei aufeinanderfolgenden Punkte (i) betrachten oder einen Punkt eliminieren und Backtracking durchführen (ii). Da die Menge $M$ aus $N$ Punkten besteht, kann sowohl Fall (i) als auch (ii) höchstens $N$-mal eintreten. Also muß höchstens $2 \cdot N$-mal die Orientierung dreier Punkte geprüft werden. Somit ist der Aufwand für die Abtastung (4) linear. Bezeichnet $f_i(N)$ $i = 1, \ldots, 5$ die Aufwandfunktion für den Schritt $i$ in Abhängigkeit von der Anzahl der Punkte $N$, so folgt für den Gesamtaufwand:

$$\begin{aligned} t &\leq c_1 \cdot f_1(N) + c_2 \cdot f_2(N) + c_3 \cdot f_3(N) + c_4 \cdot f_4(N) + c_5 \cdot f_5(N) \\ &\leq c_1 \cdot N + c_2 \cdot N + c_3 \cdot N \cdot \log N + c_4 \cdot N + c_5 \cdot N \leq c_6 \cdot N \cdot \log N \end{aligned}$$

Also benötigen wir zur Bestimmung von $SEP(M)$ einen Aufwand von $O(N \log N)$. □

Vergleichen wir nun die Laufzeit der Grahamschen Abtastung mit der unteren Schranke $\Omega(N)$, die die zur Lösung unseres Problems benötigte Zeit abschätzt, so ergibt sich, daß die Abtastung nach GRAHAM optimal ist. Somit bietet sich dieser Algorithmus nicht zuletzt auch wegen seiner einfachen Struktur für eine Implementation an.

Allerdings hat der Algorithmus auch einige Nachteile.

1. Er besitzt keine rekursive Struktur, wodurch kein Einsatz auf Parallelrechnern möglich ist. In diesem Fall ist ein Algorithmus nötig, der die Gesamtlösung des Problems durch das Lösen vieler ähnlicher Teilprobleme ermittelt.

2. Der Algorithmus arbeitet nicht online, d.h. vor der Anwendung des Algorithmus müssen alle Punkte bekannt sein, von denen die konvexe Hülle bestimmt werden soll. Möchte man beispielsweise einen Punkt $p$ aus $M$ löschen oder hinzufügen, so muß die konvexe Hülle unabhängig vom bisherigen Wissen über die alte konvexe Hülle $SEP(M)$ ermittelt werden. Die Vorinformation bleibt bei diesem Ansatz völlig unberücksichtigt, was mit einem erheblichen Mehraufwand an Zeit verbunden ist, falls man $SEP(M \setminus \{p\})$ bzw. $SEP(M \cup \{p\})$ bestimmen möchte.

3. In manchen Situationen, in denen wir die Struktur der vorliegenden Punktemenge ausnützen können, gibt es effizientere Ansätze.

Im folgenden werden Algorithmen vorgestellt, die diese Nachteile nicht mehr besitzen.

## 1.3 Der QUICKHULL-Algorithmus

In 1.1.4 haben wir gesehen, daß das allgemeine Sortierproblem eng mit dem Problem der Bestimmung von $SEP(M)$ verknüpft ist.
Es liegt demzufolge nahe, zu versuchen, einen Algorithmus, der das Sortierproblem rekursiv löst, als Grundstock für eine rekursive Lösung von $SEP(M)$ zu benutzen. Zur Sortierung von $N$ beliebigen Daten $K_1, \ldots, K_N$ verwendet man häufig den sog. QUICKSORT-Algorithmus. Dieser besitzt eine rekursive Struktur. Dem Algorithmus liegt folgende Sortieridee zugrunde: Wähle ein beliebiges Startdatum $K_i$ und bilde anschließend aus den Daten zwei Teildateien, mit $K_j \leq K_i$ für alle Sätze in der einen, und $K_j > K_i$ für alle aus der Zweiten. Wende dieses Teilungsprinzip auf beide Teildateien rekursiv an, bis nur noch einelementige Mengen übrig sind.

Wie können wir dieses Prinzip übertragen, um die Menge $SEP(M)$ zu berechnen? Dazu gehen wir auf folgende Weise vor. Wir bestimmen zunächst in einem Initialisierungsschritt den Punkt $l$ mit der kleinsten x-Koordinate sowie den Punkt $r$ mit der größten x-Koordinate (siehe Abbildung 1.4). Den beiden Teilmengen $S^{(1)}$ und $S^{(2)}$ weisen wir nun die Punkte aus $M$ so zu, daß in der einen Menge all die Punkte $p$ liegen, für die die Punkte $l, p$ und $r$ rechtsorientiert sind bzw. auf einer Geraden liegen. In die Zweite nehmen wir alle Punkte $p \in M$ auf, für die die Punkte linksorientiert sind oder auf einer Geraden liegen. (Beachte dabei, daß $p = l$ oder $p = r$ möglich ist! Folglich wird $M$ nicht disjunkt in zwei Teilmengen zerlegt, da $S^{(1)} \cap S^{(2)} \supseteq \{l, r\}$). Die Mengen $S^{(1)}$ und $S^{(2)}$ werden nun ihrerseits wieder in zwei Teilmengen $S^{(1,1)}, S^{(1,2)}$ bzw. $S^{(2,1)}$ und $S^{(2,2)}$ aufgespalten. Dazu bestimmen wir aus der jeweiligen Menge einen Punkt $h$ so, daß das Dreieck $\triangle hlr$ einen maximalen Flächeninhalt besitzt. Sollte es mehrere Punkte mit dieser Eigenschaft geben, so wählen wir als Punkt $h$ denjenigen, für den der Winkel $\angle hlr$ maximal wird. Nach unserer Konstruktion ist $h$ nun ein Punkt der konvexen Hülle, da, falls wir eine Gerade parallel zu $\overline{lr}$ durch den Punkt $h$ legen, oberhalb von dieser nach Wahl von $h$ kein weiterer Punkt mehr liegen kann, weil sonst $h$ nicht die gewünschte Maximaleigenschaft aufweisen würde. Auf dieser Gerade können noch weitere Punkte liegen. Da jedoch der Winkel $\angle hlr$ maximal gewählt wurde, ist $h$ nicht als Konvexkombination zweier Punkte darstellbar und ist somit ein Eckpunkt.

Alle Punkte, die im Inneren des Dreiecks $\angle hlr$ liegen sind nach (K1) innere Punkte. Weil wir nur Punkte aus $SEP(M)$ suchen, lassen wir sie bei weiteren

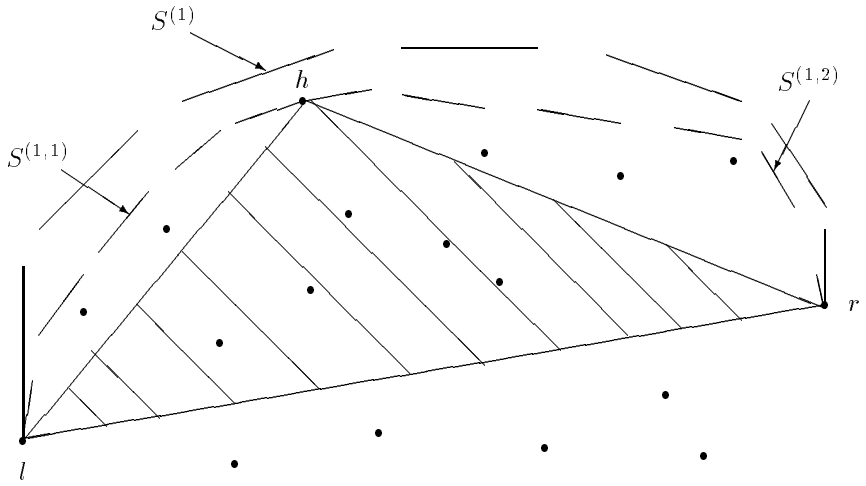



Abbildung 1.4: Mittels der Punkte $l, r$ und h wird die Menge $S^{(1)}$ geteilt und die im schraffierten Dreieck liegenden Punkte werden bei weitere Betrachtungen nicht mehr berücksichtigt.

Betrachtungen außer acht. Die Teilmenge $S^{(1,1)}$ besteht aus all denjenigen Punkten $p$ aus $S^{(1)}$, für die die Punkte $l, p$ und $h$ rechtsorientiert angeordnet sind, also den Punkten $p$, die oberhalb der Geraden $\overline{lh}$ oder auf dieser liegen. In Analogie ergibt sich die Menge $S^{(1,2)}$ aus all den Punkten $p \in S^{(1)}$, für die die Punkte $h, p$ und $r$ rechtsorientiert — in diesem Fall oberhalb der Geraden $\overline{hr}$ — bzw. auf der Gerade $\overline{hr}$ liegen. Auf die beiden Teilmengen $S^{(1,1)}$ und $S^{(1,2)}$ muß nun rekursiv diese Aufspaltungsvorschrift angewandt werden, bis die verbleibenden Mengen nur noch aus Eckpunkten bestehen. Dies ist der Fall, falls die Teilmengen nur noch zwei Punkte enthalten. Die Aufspaltungsvorschrift hängt entscheidend von den Punkten $l$ und $r$ ab. Im Fall der Aufspaltung von $S^{(1,1)}$ ist zu beachten, daß der Punkt $r$ bei weiteren Betrachtungen durch den Punkt $h$ ersetzt wird, während im Falle von $S^{(1,2)}$ der Punkt $l$ durch den Punkt $h$ ersetzt wird.

Die Menge $SEP(M)$ ergibt sich dadurch, daß die Teilmenge $S^{(2)}$ analog aufgespalten wird und anschließend die erzeugten zweielementigen Teilmengen so vereinigt werden, daß die Eckpunkte wie gewünscht sortiert angeordnet sind.

**Bemerkung:** Den Initialisierungsschritt können wir leicht auf den allgemeinen Rekursionsschritt zurückführen. Dazu ermitteln wir wie angegeben den Punkt $l = (x_0, y_0)$. Als Startpunkt $r$ wählen wir jedoch den künstlichen Punkt $r = (x_0, y_0 - \varepsilon)$ $(\varepsilon > 0)$ und spalten anschließend die Menge $M$ nach der Spaltungsvorschrift auf. Am Ende muß der ergänzte Punkt eliminiert werden.

Die rekursive Prozedur QUICKHULL sieht nun folgendermaßen aus:

```
procedure QUICKHULL (S;l,r)
begin
if (S={l,r}) then return (l,r);
h ← edge_of_biggest_triangle(S;l,r);
S[1] ← { Menge aller Punkte aus S, die entweder auf lh oder links von
lh liegen} ;
S[2] ← { Menge aller Punkte aus S, die entweder auf oder links von hr
liegen } ;
return (QUICKHULL(S[1];l,h)*QUICKHULL(S[2];h,r)\ h);
```

Hierbei ist

- edge_of_biggest_triangle eine Funktion, die den Punkt $h$ nach obiger Vorschrift bestimmt,
- * eine Funktion, die zwei Listen von Punkten durch Aneinanderhängen verknüpftund
- $\setminus$ eine Funktion, die einen Punkt aus einer Punktliste löscht.

Die Aufruffunktion für QUICKHULL hat folgende Gestalt:

```
begin
l = {x0, y0} ; /* der Punkt mit dem kleinsten Abszissenwert aus M */
r = {x0, y0 − ε} ;
SEP(M) =QUICKHULL(M; l, r);
SEP(M) = SEP(M) \ r ;
end.
```

Über den benötigten Zeitaufwand von QUICKHULL gibt nachfolgender Satz Auskunft:

**Satz 1.3.1:** *Zur Bestimmung der Menge $SEP(M)$ mittels QUICKHULL ist ein Aufwand von $O(N \log N)$ erforderlich, falls die Punkte gleichverteilt sind. Im günstigsten Fall, also falls die Menge $M$ und $SEP(M)$ übereinstimmen (V1) und bei der Aufspaltung eine der beiden Mengen $S^{(1)}$ oder $S^{(2)}$ nur genau aus zwei Eckpunkten besteht (V2), so besitzt QUICKHULL eine Laufzeit von $O(N^2)$.* ◇

*Beweis.* Die Laufzeit $t$ der rekursiven Prozedur ist proportional zum Produkt aus der Rekursionstiefe $i$ und der Laufzeit pro Rekursionsschritt $t_r$. Also gilt:

$$t \leq c \cdot i \cdot t_r \ (1).$$

Bei jedem Rekursionsschritt müssen die beiden Mengen $S^{(1)}$ und $S^{(2)}$ aufgestellt sowie eine Abfrage durchgeführt werden. Die Abfrage, ob die Menge $S$ nur

aus den Punkten $l$ und $r$ besteht, ist mit konstantem Zeitaufwand durchführbar. Die beiden Teilmengen können mit einem Aufwand von $O(N)$ bestimmt werden, da eine einmalige Betrachtung jedes Punktes $p \in S$ genügt, um diesen Punkt der entsprechenden Menge zuzuordnen, und da die Menge $S$ maximal aus $N$ Elementen bestehen kann. Also gilt:

$$t_r \leq c_1 \cdot N \ (2).$$

Setzen wir eine Gleichverteilung der Punkte in dem Sinne voraus, daß die jeweils bestimmten Teilmengen $S^{(1)}$ und $S^{(2)}$ etwa gleich viele Elemente enthalten, so hängt die Rekursionstiefe $i$ logarithmisch von $N$ ab, d.h.

$$i \leq c_2 \cdot \log N \ (3).$$

Setzen wir die Ergebnisse (2) und (3) in (1) ein, so erhalten wir:

$$t \leq c \cdot c_1 \cdot N \cdot c_2 \cdot \log N = c_3 \cdot \log N \cdot N.$$

Also benötigen wir einen Aufwand von $O(N \log N)$.
Sind die Voraussetzungen (V1) und (V2) erfüllt, so besteht bei einer Rekursionstiefe von $i$ entweder die Menge $S^{(1)}$ oder $S^{(2)}$ aus $N - i$ Elementen. Bis das Abbruchkriterium für beide Mengen erfüllt ist, haben wir also eine Rekursionstiefe von $N - 2$ erreicht. Für den Zeitaufwand gilt somit:

$$t \leq c \cdot i \cdot t_r = c \cdot (N-2) \cdot c_1 \cdot N \leq c_3 \cdot N^2.$$

Falls die Punkte in obigem Sinne günstig liegen, ist folglich ein Aufwand von $O(N^2)$ erforderlich. □

## 1.4 Der Algorithmus von OVERMARS / VAN LEEWEN

Für manche Problemstellungen ist es wünschenswert eine Algorithmus zu entwickeln, der das sogenannte ONLINE-Prinzip erfüllt. Diesem liegt nachfolgende Problemstellung zu Grunde:
Sei $S$ eine zweidimensionale Punktmenge, deren konvexe Hülle bekannt ist. Sei $(p_1, \ldots, p_N)$ eine Folge von Punkten, die in $S$ eingefügt oder aus $S$ gelöscht werden sollen. Gesucht ist ein Algorithmus, der nach Abarbeitung des i-ten Punktes $p_i$ die konvexe Hülle $co(S \cup \{p_1, \ldots, p_i\})$ ermittelt.
Ein ONLINE-Algorithmus wurde von OVERMARS und VAN LEEUWEN vorgestellt. Zur Realisierung greifen die beiden auf ein bekanntes Werkzeug aus der Informatik zurück. Sie benutzen binäre Bäume.

**Definition 1.4.1:**
Ein *binärer Baum* ist eine Menge $T = (V, E)$, wobei $V$ eine endliche Menge

von Knoten und $E$ eine Menge von Kanten ist. Die Menge der Knoten ist entweder leer oder besteht aus einem Wurzelknoten und zwei disjunkten binären Teilbäumen. Diese werden als *linke* und *rechte Teilbäume* bezeichnet. Die Wurzel des linken Teilbaums $k_l$ wird *linker Sohn*, die des Rechten $k_r$ *rechter Sohn* genannt. Die zu zwei Teilbäumen gehörende Wurzel ist der *Vater* der beiden Knoten $k_l$ und $k_r$. Ein *Blatt* des Baums ist ein Knoten, der keine Söhne besitzt. Die Wurzel des Baums ist ein Knoten, der keinen Vater hat. Die Kanten sind ungerichtet und verbinden jeden Vaterknoten mit seinen Söhnen.
Ein binärer Baum $T = (V, E)$ heißt *Suchbaum* oder *sortierter Baum*, falls es eine Abbildung $k : V \to M$, $M$ vollständig geordnet, gibt mit

$$k(v_r) \leq k(v)$$

$$k(v_l) \geq k(v)$$

für alle $v \in V, v_r \in R(v)$ und $v_l \in L(v)$. Dabei bezeichnet $R(v)$ den rechten Teilbaum von $v$ und $L(v)$ den Linken. k(v) heißt *Schlüsselwert* des Knotens $v$. Die *Stufe eines Knotens* wird rekursiv definiert. Die Stufe der Wurzel ist 1. Befindet sich ein Knoten auf Stufe $i$, so liegen seine Söhne auf Stufe $i + 1$. Die *Höhe* eines Baums ist die Zahl der Stufen minus eins. ◇

**Bemerkung:** Mit Hilfe des Schlüsselwertes $k(v)$ kann ein Knoten eindeutig identifiziert werden. Die Abbildung ist im Allgemeinen eine Projektion auf ein oder mehrere Komponenten des Knotens.

In diesen binären Bäumen werden nun Informationen über zwei Teilmengen verwaltet, deren Durchschnitt die konvexe Hülle einer endlichen Punktmenge ergibt.

**Definition 1.4.2:**
Sei $M$ eine endliche Punktmenge des $\mathbb{R}^2$. Sei $u_\infty = (0, -\infty)$ und $l_\infty = (0, +\infty)$. So bezeichnet man $co(M \cup u_\infty)$ als *U-Hülle* (upper-hull) und $co(M \cup l_\infty)$ als *L-Hülle* (lower-hull). ◇

Diese Definition veranschaulicht Diagramm 1.5. Hierbei sind die Punkte willkürlich durchnummeriert. Zur Abspeicherung der Information über die U-Hülle — für die L-Hülle geht dies analog— werden höhenbalancierte binäre Bäume verwendet, da bei diesen Einfüge- und Löschoperationen sehr effektiv durchgeführt werden können. Dies geht mit einem Aufwand, der logarithmisch von der Gesamtzahl aller Daten abhängt (vgl. [22]). Zur Realisierung einer geeigneten Datenstruktur am Rechner kann man zum Beispiel AVL-Bäume (Addelson, Velski, Landis; 1962) benutzen. Mit den nachfolgend aufgeführten Funktionen — eine Beschreibung findet man in [10]— werden die Grundoperationen auf AVL-Bäumen umgesetzt. Die Arbeitsweise dieser Funktionen ist für das Verständnis des Algorithmus von OVERMARS / VAN LEEUWEN nicht erforderlich. Die

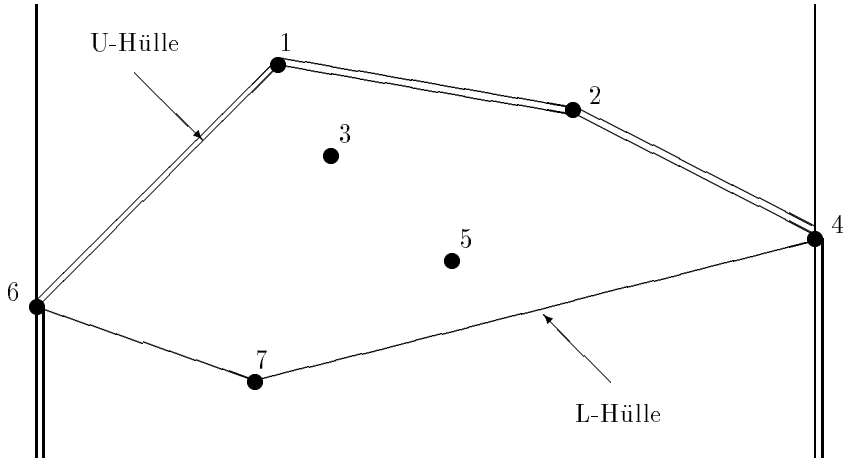



Abbildung 1.5: U-Hülle und L-Hülle einer Punktmenge

angegebenen Laufzeiten spielen jedoch bei späteren Laufzeitbetrachtungen eine wichtige Rolle.

| **Funktionsname** | **Aufgabe** | **Laufzeit** |
|---|---|---|
| AVL_SUCHEN | Suchen eines vorg. Wertes | $O(\log N)$ |
| AVL_INSERT | Einfügen eines Wertes | $O(\log N)$ |
| AVL_DELETE | Löschen eines Wertes | $O(\log N)$ |

Der interessierte Leser findet eine Beschreibung der Funktionen in [22].

**Definition 1.4.3:**
Gegeben sei ein binärer Baum $T$.
Dann heißt

$$b(v) = \begin{cases} 0 & \text{, falls } v \text{ ein Blatt ist,} \\ h(r(v)) + 1 & \text{, falls } l(v) = \emptyset,\ r(v) \neq \emptyset, \\ -h(l(v)) + 1 & \text{, falls } l(v) \neq \emptyset,\ r(v) = \emptyset, \\ h(l(v)) - h(r(v)) & \text{, sonst,} \end{cases}$$

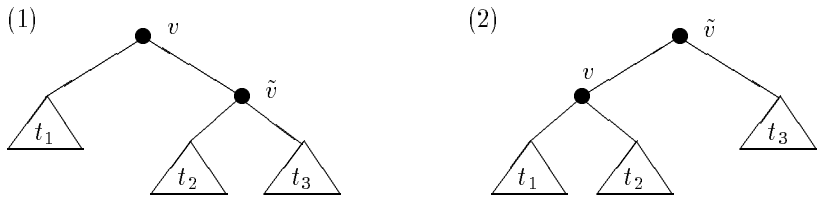
*Balance* eines Knotens $v$. Hierbei bezeichnet $r(v)$ den rechten Teilbaum von $v$, $l(v)$ den linken Teilbaum und $h(\tilde{T})$ die Höhe eines Teilbaums $\tilde{T}$. Ein binärer Baum $T$ heißt *ausgeglichen* oder *AVL-Baum*, falls gilt:

$$|b(v)| \leq 1 \; \forall v \in V.$$

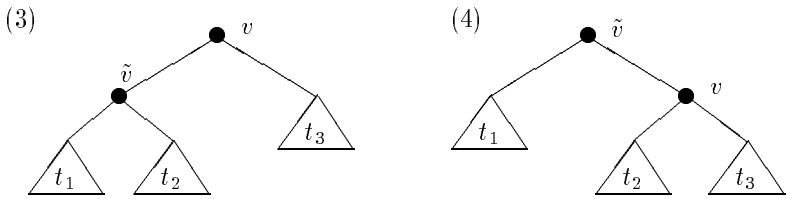
$\diamond$

Ist die Balance-Eigenschaft eines Knotens im AVL-Baum verletzt, so kann diese mit Hilfe einer Folge von Rotationen wiederhergestellt werden.

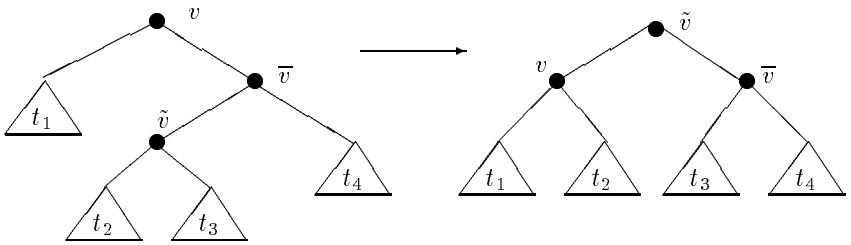
**Definition 1.4.4:**
Gegeben sei ein Suchbaum $T = (V, E)$ und $v, \tilde{v}, \overline{v} \in V$. Ferner seien $t_1, t_2, t_3$ und $t_4$ Teilbäume. $v, \tilde{v}, t_1, t_2$ und $t_3$ seien wie in (1) angeordnet:



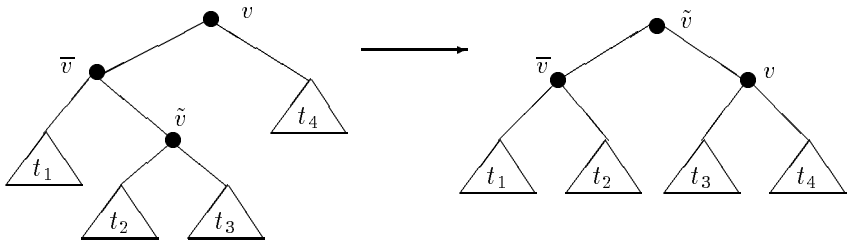

Dann bezeichnen wir eine Umordnung der Knoten in der Art, daß wir eine Situation wie in (2) erhalten, als *Linksrotation* im Baum $T$.




Seien die Knoten dagegen wie in (3) angeordnet, und wir erhalten nach dem Umordnen eine Situation wie in (4), dann bezeichnen wir diesen Vorgang als *Rechtsrotation* in $T$.
Eine Transformation der Form




heißt *Linksdoppelrotation*.
Werden die Knoten wie folgt umgeordnet,




so bezeichnen wir diese Operation als *Rechtsdoppelrotation*. ◇

Der nachfolgende Satz stellt sicher, daß mit Hilfe einer Folge von Einfach- und Doppelrotationen Balanceverletzungen beseitigt werden können, bei denen für einen Knoten $v$ gilt: $|b(v)| = 2$. Dies tritt beim Einfügen oder Löschen eines Knotens häufig auf.

**Satz 1.4.5:** *Sei $T = (V, E)$ ein binärer Baum mit der Wurzel $v \in V$. Falls der Knoten $v$ die Bedingung $|b(v)| = 2$ erfüllt, und die Teilbäume $r(v)$ und $l(v)$ AVL-Bäume sind, so kann $T$ mittels einer Rotation oder Doppelrotation in einen AVL-Baum $\tilde{T}$ übergeführt werden.* ◇

*Beweis.* Den Beweis findet der Leser in [22]. □

Weist die Baumstruktur eine derartige Verletzung auf, so startet man bei dem Knoten $v$, der nicht die richtige Balance besitzt. Mit Hilfe geeigneter Rotationen, die aus dem Beweis des vorangehenden Satzes ersichtlich sind, beseitigt man diese Verletzung. Nun überprüft man die Balance des Vaterknotens von $v$. Ist sie ebenfalls fehlerhaft, so kann nur gelten: $|b(VATER(v))| = 2$. Dies kann man durch entsprechende Rotationen ausgleichen. Im Anschluß daran betrachtet man dessen Vater, auch wenn die Balance jenes Knotens richtig war. Auf diese Weise arbeitet man sich sukzessive bis zum Wurzelknoten vor. Dort kann endgültig die Balanceverletzung aufgehoben werden und man erhält einen AVL-Baum. Zur Realisierung der Knoten eines Baums benötigen wir eine Struktur KNOTEN. Diese besteht aus folgenden Komponenten:

- **LSOHN**, ein Zeiger auf ein Element vom Typ KNOTEN, welches den linken Sohn repräsentiert.
- **RSOHN**, ein Zeiger auf ein Element vom Typ KNOTEN, das den rechten Sohn darstellt.
- **BAUM**, ein Zeiger auf die Wurzel eines AVL-Baums bzw. im Falle eines Blattes ein Zeiger auf ein Element vom Typ KOORDINATEN. Dieses besteht aus zwei Komponenten, in denen die x- und y-Koordinaten eines Punktes aus der Ebene gespeichert wird.

- **STUFE**, ein Integerwert, der die Stufe angibt, auf der sich ein Knoten befindet.
- **J**, ein Integerwert, der für die Bestimmung von U-Hüllen erforderlich ist.

Haben wir einen Knoten $v$, also ein Element vom Typ KNOTEN, so bezeichnen wir dessen Komponenten mit LSOHN($v$), RSOHN($v$), Baum($v$), Stufe($v$) und J($v$). Bei der Verwaltung der Daten gelten die folgenden Grundsätze:

**(P1)** Die Information über die Koordinaten der einzelnen Punkte wird grundsätzlich nur in den Blättern eines AVL-Baums $T$ gespeichert. Dort werden die Punkte nach ihrem Abszissenwert sortiert abgelegt. Bei identischer x-Koordinate werden diese Punkte sortiert nach dem Ordinatenwert hinterlegt. Die Information über den Punkt mit der kleinsten x-Koordinate wird im äußersten linken Blatt gespeichert.

**(P2)** Im Wurzelknoten — auf diesen weist der Zeiger *wurzel* — wird ein Zeiger auf einen ausgeglichenen sortierten Baum hinterlegt. In diesem Baum werden Zeiger auf Elemente vom Typ KOORDINATEN gespeichert. Jeder zeigt auf ein Element, dem die Koordinaten eines Eckpunktes der U-Hülle aller Punkte zugewiesen worden sind. Alle Zeiger charakterisieren die U-Hülle. Darüber hinaus wird den inneren Knoten — solche die keine Blätter sind — ebenfalls ein Zeiger auf einen sortierten AVL-Baum zugeordnet. Wir streben nun an, in dem Baum $T$ so Information bereitzustellen, daß wir zu jedem Knoten $v$, den wir als Wurzelknoten eines Teilbaums interpretieren können, jeweils die U-Hülle der Punkte bestimmen können, die in den zugehörigen Blättern gespeichert sind. Die Menge dieser Eckpunkte bezeichnen wir mit $U(v)$.
Beispielsweise könnten wir dazu Zeiger auf Elemente vom Typ KOORDINATEN in ausgeglichenen sortierten Bäumen ablegen, wobei in jedem Element die Koordinaten eines Punktes gespeichert sind, der ein Eckpunkt der jeweiligen U-Hülle ist. In diesem Fall wird jedoch redundante Information gespeichert. Wir müssen beachten, daß manche Punkte sowohl in $U(v)$ als auch in $U(VATER(v))$ vorkommen. Der benötigte Speicherplatz läßt sich folglich dadurch reduzieren, indem wir in dem Baum auf den Baum($v$) zeigt, nur die Zeiger ablegen, deren zugehörige Punkte in $U(v)$ aber nicht mehr in $U(VATER(v))$ liegen.

**Bemerkung:**

- Zur Veranschaulichung des Algorithmus numerieren wir die Punkte aus der Menge $M$. Die Nummer des Punktes entspreche dem Zeiger auf das Element, in dem die Information über die Koordinaten des Punktes gespeichert sind. In der Praxis ist der Zeiger nicht die Nummer des Punktes, sondern die Speicheradresse, an der das Element vom Typ KOORDINATEN hinterlegt ist.

- Die Zeiger auf die Elemente vom Typ KOORDINATEN sind die Schlüsselwerte der Blätter. Um zwei Zeiger $z_1, z_2$ miteinander vergleichen zu können, führen wir folgende Ordnung ein:
  - $z_1 = z_2$ genau dann, wenn die Punkte, deren Koordinaten in Elementen vom Typ KOORDINATEN gespeichert sind, auf welche $z_1$ und $z_2$ zeigen, in x- und y-Koordinate übereinstimmen.
  - $z_1 < z_2$, falls der Punkt $p$, auf den $z_1$ zeigt, eine kleinere x-Koordinate besitzt als der Punkt $q$, der durch $z_2$ repräsentiert wird. Haben die beiden Punkte dieselbe x-Koordinate, so ist die y-Koordinate des Punktes $p$ kleiner.
- Da wir die $k$ Elemente, die in dem binären sortierten Baum abgespeichert sind, wie Elemente eines Arrays indizieren können, stellen wir die Einträge in Form einer Liste $L = (e_1, \ldots, e_k)$ dar. Die Numerierung erhalten wir mit Hilfe eines *postorder-Durchlaufs*. Dieser wird durch folgende rekursive Prozedur realisiert. Dabei beginnen wir beim Wurzelknoten $wurzel$:

```
procedure postorder(v)
if (v==Blatt) then return;
postorder(LEFT(v));
Numeriere den Knoten v;
postorder(RIGHT(v));
return;
```

Darüber hinaus brauchen wir noch drei Funktionen:

1. AVL_VERKNÜPFE$(z_1, z_2)$
   $z_1$ und $z_2$ seien Zeiger auf zwei sortierte AVL-Bäume $T_1$ und $T_2$. Für beliebiges $k \in T_1$ und $l \in T_2$ gelte: $k \leq l$. AVL_VERKNÜPFE$(z_1, z_2)$ soll einen Zeiger auf einen sortierten AVL-Baum liefern, in dem alle Elemente aus $T_1$ und $T_2$ eingetragen sind.
2. AVL_AUFSPALTEN$(u, z)$
   $z$ sei ein Zeiger auf einen sortierten AVL-Baum $T$.
   $u$ sei ein Zeiger auf ein Element, das mit den in dem AVL-Baum eingetragenen Elementen verglichen werden kann.
   AVL_AUFSPALTEN$(u, z)$ soll zwei Zeiger $z_1$ und $z_2$ liefern, die auf zwei AVL-Bäume $T_1$ bzw. $T_2$ weisen. Für alle Elemente $v \in T_1$ soll gelten: $v \leq u$. $T_2$ enthalte die Elemente aus $T$, die in $T_1$ nicht eingetragen sind.
3. BRÜCKE$(z_1, z_2)$
   $z_1$ und $z_2$ seien Zeiger auf sortierte AVL-Bäume $T_1$ und $T_2$, in denen Zeiger auf Elemente vom Typ KOORDINATEN gespeichert sind. Die

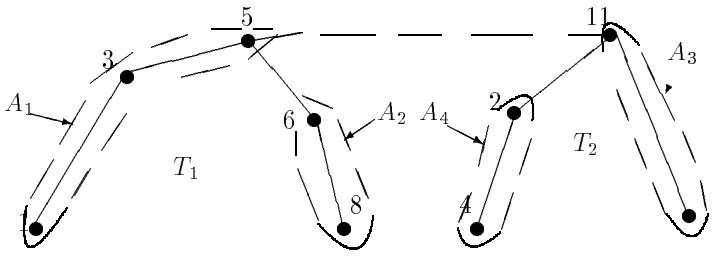



Abbildung 1.6: Veranschaulichung der Punktmengen, die von der Funktion BRÜCKE bestimmt werden sollen

> Punkte, welche in den Elementen gespeichert sind, auf die die Zeiger der Bäume weisen, seien die Eckpunkte zweier U-Hüllen. Bezeichnen wir mit $max$ den größten x-Koordinatenwert der Punkte aus $T_1$ und mit $min$ den kleinsten Abszissenwert der Punkte aus $T_2$, so gelte: $max \leq min$. BRÜCKE$(z_1, z_2)$ soll nun vier Zeiger auf sortierte AVL-Bäume $A_1, A_2, A_3$ und $A_4$ liefern. In $A_1$ werden all diejenigen Zeiger gespeichert, die auf Punkte zeigen, welche in der U-Hülle der Punkte aus $T_1$ sowie in der Hülle der Punkte aus $T_1 \cup T_2$ vorkommen. $A_2$ enthalte alle diejenigen Zeiger, welche auf Punkte zeigen, die in der U-Hülle von $T_1$ enthalten sind, aber nicht mehr in der der Punkte aus $T_1 \cup T_2$. $A_3$ und $A_4$ ergeben sich analog zu $A_1$ und $A_2$, wobei statt Punkte aus $T_1$ Elemente aus $T_2$ betrachtet werden. Die Problemstellung veranschaulicht Diagramm 1.6.

In ([10], S. 466-469) ist eine Lösung von C. A. Crane für die Umsetzung der Funktionen AVL_VERKNÜPFEN und AVL_AUFSPALTEN beschrieben. Details zur Realisierung der Funktion BRÜCKE findet man in [13] auf den Seiten 126f.

Wir wollen nachfolgend darstellen, wie die Koordinaten eines beliebigen Punktes $p \in \mathbb{R}^2$ in einen AVL-Baum $T$ eingefügt werden, in dem nach dem angegebenen Konzept Information über die U-Hülle der Punkte $p_1, \ldots, p_N$ gespeichert ist. Nach dem Einfügen soll der AVL-Baum $\tilde{T}$ so aufgebaut sein, daß zum einen die Grundsätze (P1) und (P2) für $\tilde{T}$ weiterhin gültig sind, und zum anderen im Wurzelknoten die Information über die U-Hülle der Punkte $p_1, \ldots, p_N, p$ ablesbar ist. Dazu müssen wir folgende Schritte durchführen:

1. Ermittlung des Einfügepfades.
2. Einfügen eines neuen Knotens.
3. Ausgleichsoperationen in entgegengesetzter Richtung entlang des Einfügepfades; Update der U-Hüllen-Information in den Knoten.

Realisierung von (1):

Der zu $p$ gehörende Einfügepfad besteht aus einer Folge von Knoten $\{k_1, \ldots, k_s\}$ aus dem AVL-Baum $T$ mit nachfolgenden Eigenschaften:

- $k_1$ ist der Wurzelknoten,
- $k_s$ ist ein Blatt und
  - alle Punkte, die in Blättern links von $k_s$ gespeichert sind, besitzen einen kleineren Abszissenwert als der Punkt $p$,
  - alle Punkte, die in Blättern rechts von $k_s$ gespeichert sind, haben eine eine größere x-Koordinate als der Punkt $p$,
- $k_{i+1}$ ist ein Sohn von $k_i$, $i = 1, \ldots, s-1$.

Der Einfügepfad wird iterativ beginnend mit dem Wurzelknoten aufgebaut. Um zu entscheiden, ob der nächste Knoten der linke oder rechte Sohn ist, verzweigt man ausgehend von einem bestimmten Knoten $k_j$ in dessen rechten Teilbaum und ermittelt dort die Koordinaten des Punktes, der in dem Blatt außen links gespeichert ist. Ist dessen x-Koordinate größer als die des einzufügenden Punktes, so ist der linke Sohn der nachfolgende Knoten $k_{j+1}$ im Einfügepfad, ansonsten der Rechte.

Wie wir im Schritt (3) sehen werden, ist es in Hinblick auf Ausgleichsoperationen erforderlich zu jedem Knoten $k_j$ die Menge $U(BRUDER(k_{j+1}))$ zu ermitteln und zu speichern. Kennen wir $U(k_j)$, so können wir $U(k_{j+1})$ sowie $U(BRUDER(k_{j+1}))$ mit Hilfe der Funktion AVL_AUFSPALTEN und dem Wert $J(k_j)$ ermitteln.
Eine iterative Funktion zur Lösung des Problems sieht vereinfacht wie folgt aus:

```
procedure ermittele_Einfügepfad (v,p);
 begin
   if (v ≠ BLATT) then
   begin
     (Q_L ,Q_R):=AVL_AUFSPALTEN(U(v),J(v));
     U(LSOHN(v)):=AVL_VERKNUEPFE(Q_L ,Baum(LSOHN(v)));
     U(RSOHN(v)):=AVL_VERKNUEPFE(Baum(RSOHN(v)),Q_R);
     if ( x(p) ≤ x[v] ) then
       v := LSOHN(v)
     else
       v := RSOHN(v);
     ermittele_Einfügepfad(v,p);
   end;
 end;
```

Hierbei bezeichnet x(p) die x-Koordinate des Punktes p und x[v] den Abszissenwert des Punktes, der im rechten Teilbaum im links außen stehenden Blatt gespeichert ist. Diese Routine ermittelt den vollständigen Einfügepfad, falls der Wurzelknoten als Iterationsstart benutzt wird.

In der Praxis definiert man eine Struktur, um die anfallende Information geeignet zu speichern. Sie besitzt vier Komponenten:

- **k**, ein Zeiger auf den jeweils aktuellen Knoten v,
- **richtung**, eine Variable die Auskunft darüber gibt, ob der aktuelle Knoten v linker oder rechter Sohn seines Vorgängers ist,
- **uL**, ein Zeiger auf einen AVL-Baum, in dem die Information über die Menge $U(LSOHN(v))$ gespeichert ist,
- **uR**, ein Zeiger auf einen AVL-Baum, in dem die Information über die Menge $U(RSOHN(v))$ gespeichert ist.

Für jeden Knoten des Pfades wird ein Strukturelement initialisiert. Durch die Einführung dieser Struktur verliert das Programm jedoch an Lesbarkeit. Daher geben wir das Programm vereinfacht an.

Realisierung von (2):

Wurde der Einfügepfad $k_1, \ldots, k_s$ $(s \in \mathbb{N})$ ermittelt, so kann der Punkt $p$ in den AVL-Baum $T$ eingefügt werden. Dazu werden die Koordinaten in einem neu erzeugten Element $\overline{p}$ vom Typ KNOTEN gespeichert. Üblicherweise wird der neue Knoten an den Knoten $k_s$ als Sohn angehängt werden. Anschließend muß unter Umständen eine Folge von Einfach- und Doppelrotationen durchgeführt werden, um die AVL-Baum-Eigenschaft wiederherzustellen. In unserem Fall ist dieses Vorgehen nicht sinnvoll. Denn der Knoten $k_s$ wird bei dieser Vorgehensweise zu einem inneren Knoten, weil $k_s$ der Vaterknoten des neu erzeugten Knotens wird. Damit ist in einem inneren Knoten Information über die Koordinaten eines Punktes gespeichert. Dies ist nach (P1) nicht erlaubt. Es gibt aber keine Ausgleichsoperation, die diese Verletzung beseitigen könnte, weil durch eine Rotation nur ein anderes Blatt zu einem inneren Knoten gemacht werden könnte. Wir müssen beim Einfügen wie folgt vorgehen. An den Vaterknoten von $k_s$ wird ein neu erzeugter innerer Knoten $\tilde{p}$ angehängt. Dieser erhält die Knoten $k_s$ und $\overline{p}$ als Söhne. Ist der Abszissenwert des Punktes, der im Knoten $k_s$ gespeichert ist, kleiner als derjenige von $\overline{p}$, so wird $k_s$ der linke Sohn von $\tilde{p}$ ansonsten $\overline{p}$. Sind beide identisch, so wird der Knoten, dem der Punkt mit der kleineren y-Koordinate zugeordnet ist, der linke Sohn von $\tilde{p}$. Bei Übereinstimmung beider Koordinaten ist eine Fehlermeldung auszugeben und der Knoten $\overline{p}$ darf nicht in $T$ eingefügt werden.

Realisierung von (3):

Unser Ziel ist es nun durch geeignete Ausgleichsoperationen einen AVL-Baum herzustellen, der den Regel P1 und P2 genügt. Wir betrachten dazu den Fall, daß beim Rückwärtsdurchlauf des Pfades keine Ausgleichsoperationen in Form von Einfach- und Doppelrotationen aufgrund einer Verletzung der Balance erforderlich sind. Dies genügt zum Verständnis des Strukturupdatens. In diesem Fall ist klar, daß die Regel P2 nur von Knoten entlang des Einfügepfades verletzt werden kann. Es müssen folglich für jeden Knoten, der auf dem bestimmten Einfügepfad liegt, die Information $J(v)$ und $Baum(v)$ aktualisiert werden. Dazu wird diese Knotenmenge rückwärts durchlaufen, wobei wir folgendes Problem haben: Die Menge $U(k_{j+1})$ ist uns bekannt, und die Menge $U(k_j)$ soll bestimmt werden. Dies ist mit der Funktion BRÜCKE möglich, falls die Menge $U(BRUDER(k_{j+1}))$ bekannt ist. Diese Information haben wir bei der Ermittlung des Einfügepfades notiert. Mittels der Routine BRÜCKE wird $U(k_{j+1})$ in Teile $Q_1$ und $Q_2$ und analog $U(BRUDER(k_{j+1}))$ in $Q_3$ und $Q_4$ aufgespalten. Eine Verknüpfung der Mengen $Q_1$ und $Q_4$ ergibt die Menge $U(VATER(k_{j+1})) = U(k_j)$. Die Mengen $Q_2$ und $Q_3$ enthalten die Information über die Punkte, die in $U(k_{j+1})$ bzw. $U(BRUDER(k_{j+1}))$ enthalten sind, aber nicht mehr in $U(k_j)$. $Q_2$ muß also in $Baum(LSOHN(VATER(k_{j+1})))$ und $Q_3$ in $Baum(RSOHN(VATER(k_{j+1})))$ gespeichert werden. Damit die gesuchten Mengen auf diese Weise iterativ bestimmt werden können, beginnen wir die Iteration mit der Menge $U(VATER(\overline{p}))$. Diese Menge besteht entweder aus den beiden Punkten, die in $\overline{p}$ und $k_s$ gespeichert sind, falls die x-Koordinaten verschieden sind, oder nur aus dem Punkt, welcher die größere y-Koordinate aufweist. Es ist zu beachten, daß die Mengen $U(v), Q_1, Q_2, Q_3$ und $Q_4$ intern durch Zeiger auf AVL-Bäume repräsentiert werden. Als iterative Prozedur erhalten wir:

```
procedure ausgleich (v);
   begin
      if v ≠ WURZEL then
         begin
         (Q1, Q2, Q3, Q4, J) :=BRÜCKE(U(v), U(BRUDER(v)));
         Baum(LSOHN(VATER(v))) = Q2;
         Baum(RSOHN(VATER(v))) = Q3;
         U(VATER(v))=AVL_VERKNÜPFE(Q1, Q4);
         J(VATER(v))=J;
         ausgleich (VATER(v));
         end;
      else
         Baum(v)=U(v);
   end;
```

Für die Funktionen *ermittele_Einfügepfad* und *ausgleich* ist jeweils ein Zeitaufwand erforderlich, der proportional zum Produkt aus der Höhe des Baums und dem Aufwand pro Iterationsschritt ist. Für die Höhe des AVL-Baums $T$ gilt nach [22]:

$$h(T) \leq 2 \cdot \log anz \qquad (1).$$

Hierbei bezeichnet $anz$ die Anzahl der Knoten des AVL-Baums. Zwischen der Knotenanzahl $anz$ und der Anzahl der in den Blättern gespeicherten Punkte $N$ gilt:

$$anz = 2 \cdot N - 1 \qquad (2).$$

Wir erhalten für $N \geq 2$ indem wir (2) in (1) einsetzen:

$$h(T) \leq 4 \cdot \log N \qquad (3).$$

Für den benötigten Zeitaufwand eines Iterationsschritts gilt bei der Funktion *ausgleich*

$$t \leq c + c_1 \cdot \log N + c_2 \cdot \log N \leq c_3 \cdot \log N \qquad (4).$$

da für die Abarbeitung der Funktion BRÜCKE und AVL_VERKNÜPFE jeweils ein Aufwand von $O(\log N)$ anfällt. Es gilt somit vermöge (3) und (4):

$$t_{ausgleich} \leq c_4 \cdot \log^2 N.$$

Bei der Funktion *ermittele_Einfügepfad* ist der Zeitaufwand bei der Durchführung eines Iterationsschritts konstant, falls der Wert $x[v]$ zu jedem Knoten gespeichert ist. Ist dies nicht der Fall, so ist der Aufwand zur Bestimmung von $x[v]$ abhängig von der Höhe des Baums. Wir benötigen folglich maximal $O(\log N)$ Zeit, und erhalten insgesamt:

$$t_{ermittele} \leq c_5 \cdot \log^2 N.$$

Da zum Erzeugen und Einfügen eines Knotens wie in Schritt (2) beschrieben ein konstanter Zeitaufwand $t_{einfüge}$ erforderlich ist, ergibt sich:

$$t \leq t_{ermittele} + t_{einfüge} + t_{ausgleich} \leq c_6 \cdot \log^2 N.$$

Da das Entfernen eines Knotens vergleichbare Operationen wie das Einfügen erfordert, kann man analog zeigen:

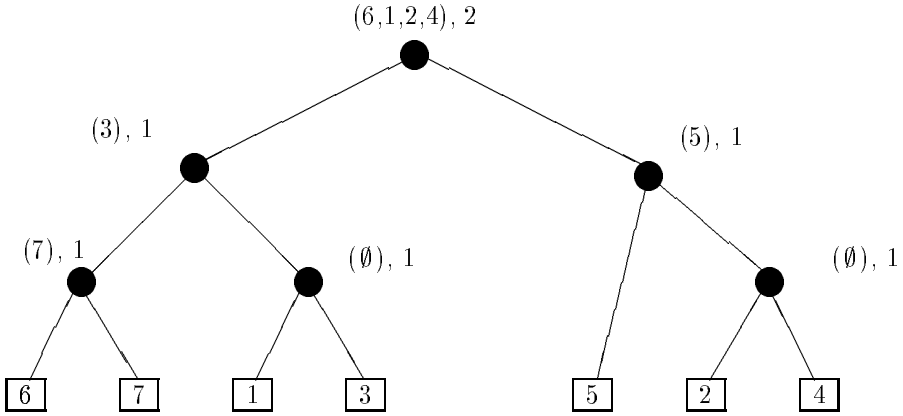
$$t_{löschen} \leq c_7 \cdot \log^2 N.$$

Zusammenfassend gilt somit:

**Satz 1.4.6:** *Die U-Hülle und L-Hülle einer Menge von $N$ Punkten aus der Ebene kann dynamisch — also dem ONLINE-Prinzip genügend — mit einem Aufwand von $O(\log^2 N)$ pro Einfüge- und Löschvorgang ermittelt werden.*

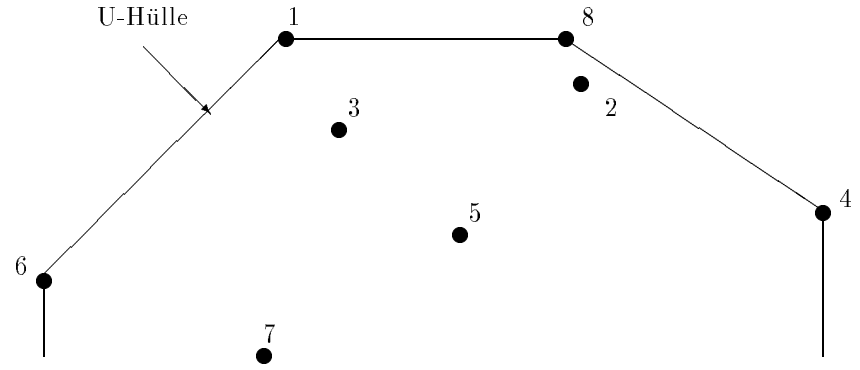
**Bemerkung:** Wollen wir mit diesem ONLINE-Algorithmus die konvexe Hülle von $N$ Punkten aus der Ebene bestimmen, so ist hierfür ein Aufwand von $O(N \log^2 N)$ nötig, da zum Lösen dieses Problems $N$ Einfügeoperationen erforderlich sind. Wir bezahlen die Fähigkeit des Algorithmus, online zu arbeiten, mit einem höheren Zeitaufwand.

**Beispiel 1.4.7:** Betrachten wir die Punkte mit Nummer $1, \ldots, 7$ aus Abbildung 1.5, so sieht der binäre Baum $T_1$, in dem die Information über die U-Hülle nach der Idee von OVERMARS/VAN LEEUWEN gespeichert ist, folgendermaßen aus:



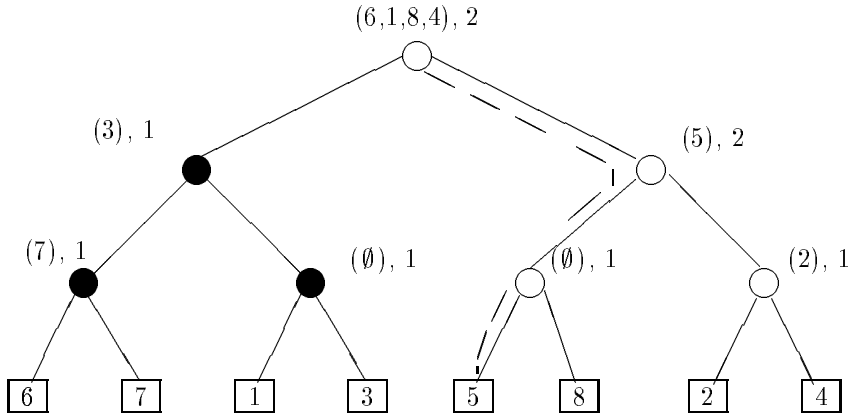

Dabei repräsentiert die nicht in Klammern stehende Zahl den Integerwert $J(v)$.

Betrachten wir nun die folgende Punktemenge, die sich aus der Menge aus 1.5 durch Hinzufügen des Punktes mit Nummer 8 ergibt:




Möchten wir aus $T_1$ den binären Baum ableiten, der die Information über die U-Hülle der Punkte mit Nummer $1, \ldots, 8$ enthält, so ergibt sich nachfolgender

Baum $T_2$ :




Die gestrichelte Linie zeigt den Einfügepfad. Nicht ausgefüllte Kreise repräsentieren Knoten, deren Inhalt aufgrund der Ergänzung des Knotens mit Nummer 8 verändert wurde. ◇

## 1.5 Der Algorithmus von AKL

In manchen Fällen kann man die Struktur einer gegebenen Punktmenge ausnützen, um einen Algorithmus zu entwickeln, für den eine bessere obere Schranke angegeben werden kann als $O(N \log N)$.

Für den Fall, daß wir die konvexe Hülle einer Punktmenge $M = \{p_1, \ldots, p_N\}$ bestimmen wollen, bei der wir von Anfang an wissen, daß die Anzahl ihrer inneren Punkte wesentlich höher als die der Eckpunkte ist, hat AKL eine effiziente direkte Methode entwickelt. In einem ersten Schritt wird der lexikographisch kleinste Punkt $p_0$ bestimmt, d.h. der Punkt, der bei der kleinsten y-Koordinate die kleinste x-Koordinate besitzt. $p_0$ ist ein Eckpunkt und dient als Start für eine sukzessive Bestimmung benachbarter Eckpunkte $\{p_1, \ldots, p_h\}$. Um ausgehend von einem Eckpunkt $p_i$ den nächsten Punkt $p_{i+1}$ festlegen zu können, verwenden wir folgendes anschauliche Kriterium:

**Kriterium 4(K4):**
Sei $p \in M$ der lexikographisch kleinste Punkt und $q \in M$ der lexikographisch größte Punkt. Sei $e$ ein Eckpunkt, der rechts der Verbindungsgeraden $\overline{pq}$ liegt. Dann ist $f$ genau dann der benachbarte Eckpunkt mit der größeren y-Koordinate, falls der Winkel, den die Verbindungsgerade $\overline{ef}$ mit der positiven x-Achse einschließt, in dem Sinne minimal ist, daß es keinen weiteren Punkt $g$ gibt, mit der Eigenschaft, daß der Winkel, den die Verbindungsgerade $\overline{eg}$ mit der positiven x-Achse einschließt, kleiner ist.
Sei $e$ ein Eckpunkt der links der Verbindungsgerade $\overline{pq}$ liegt. In diesem Fall ist $f$ genau dann der benachbarte Eckpunkt mit der kleineren y-Koordinate, wenn obige Aussagen erfüllt sind, wobei statt der positiven x-Achse die negative x-Achse betrachtet wird.

Nach Bestimmung von $p_0$ wendet man den ersten Teil des Kriteriums 4 sukzessive an, um von diesem ausgehend alle Eckpunkte zu ermitteln, die rechts der Gerade zwischen dem lexikographisch kleinsten und größten Punkt liegen. Der letzte Punkt, den man auf diese Weise bestimmt, ist der lexikographisch größte Punkt. Sobald dieser erreicht ist — diesen erkennt man z.B. dadurch, daß alle nach Kriterium 4 bestimmten Winkel größer als 180 Grad sind — legt man mittels das zweiten Teils alle Eckpunkte fest, die links von der Verbindungsgerade liegen.
Die Vorgehensweise veranschaulicht Abbildung 1.7. Ist ein Eckpunkt bekannt,

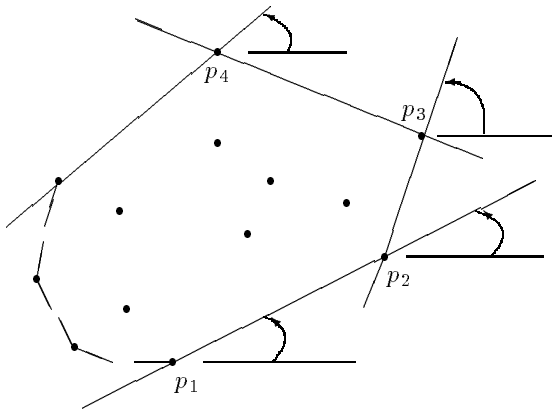



Abbildung 1.7: Der Ansatz von AKL zur Bestimmung der konvexen Hülle. Jeder Eckpunkt kann mit einem Aufwand von $O(N)$ bestimmt werden.

so benötigen wir einen Aufwand von $O(N)$ zur Bestimmung eines neuen Eckpunktes, da die Anzahl der aufzustellenden Winkel linear von der Punktzahl abhängig ist. Hat die Punktmenge $M$ $h$ Eckpunkte ergibt sich somit folgende Laufzeitaussage:

**Satz 1.5.1:** *Zur Bestimmung der Menge $SEP(M)$ mittels der Algorithmus von AKL benötigen wir einen Zeitaufwand von $O(h \cdot N)$.* $\diamond$

## 1.6 Ein Ansatz zur Approximation konvexer Hüllen

Für Anwendungen, in denen es nicht so sehr auf die Genauigkeit sondern vielmehr auf die Geschwindigkeit ankommt, können wir Algorithmen angeben, die die konvexe Hülle einer endlichen Anzahl von Punkten $M = \{p_1, \ldots, p_N\}$ approximieren. Sie finden bei statistischen Problemen Verwendung. Häufig liefern z.B. Messungen Resultate, die aufgrund der Meßungenauigkeit nicht exakt sein können. Die Ergebnisse liegen vielmehr mit einer wohldefinierten Genauigkeit vor. Möchten wir möglichst schnell ein Bild von der konvexen Hülle der Beobachtungspunkte erhalten, so können wir auf einen Algorithmus von BENTLEY-FAUST-PREPARATA (1982) zurückgreifen. Es müssen folgende Arbeitsschritte durchgeführt werden:

1. Bestimme die Punkte $m_1, \ldots, m_l$ mit der kleinsten x-Koordinate. Ist $l > 2$, so weisen wir $m_1$ und $m_2$ unter den bisher bestimmten Punkten die beiden Punkte mit der größten bzw. kleinsten y-Koordinate zu und setzen $l = 2$.

2. Ermittle alle Punkte $q_1, \ldots, q_r$, die die größte x-Koordinate besitzen. Ist $r > 2$, so ordne $q_1$ und $q_2$ die beiden Punkte mit extremaler y-Koordinate zu und setze $r = 2$.

3. Unterteile den Streifen, der durch die vertikalen Geraden durch die Punkte $m_1, \ldots, m_l$ und $q_1, \ldots, q_r$ festgelegt wird, in $k$ gleichbreite Streifen $K_1, \ldots, K_k$.

4. Ordne alle Punkte $p \in M \setminus \{m_1, \ldots m_l, q_1, \ldots, q_r\}$ dem Streifen $K_i$ zu, in dem sie liegen. Die Nummer $i$ des Streifens ergibt sich nach folgender Formel:
$$i = int(k \cdot (x - xmin)/(xmax - xmin)) + 1.$$
Hierbei ist

   - $int(x)$ Funktion, die die ganze Zahl bestimmt, die sich durch Abschneiden der Nachkommastellen von $x$ ergibt,

- $x$ x-Koordinate des Punktes $p$,
- $xmax$ x-Koordinate der Punkte $q_1, \ldots, q_r$ und
- $xmin$ x-Koordinate der Punkte $m_1, \ldots, m_l$.

5. Ermittle in jedem Streifen $K_i$ den Punkt $p_{i1}$, der die größte y-Koordinate aufweist, und den Punkt $p_{i2}$ mit der kleinsten y-Koordinate. $p_{i1}$ und $p_{i2}$ können identisch sein, falls alle Punkte des Streifens auf einer horizontalen Geraden liegen. Enthält der Streifen keine Punkte, so können diese beiden Punkte nicht bestimmt werden. Dies ist jedoch für den Gesamtablauf des Algorithmus nicht entscheidend.

6. Bestimme die konvexe Hülle der Punkte

$$S := \{p_1, \ldots, p_l, q_1, \ldots, q_r, p_{11}, p_{12}, \ldots, p_{k1}, p_{k2}\}.$$

Die Menge $co(S)$ ist in der Regel nur ein Approximation für die Menge $co(M)$. Die Genauigkeit der Approximation ist vom Parameter $k$ abhängig. Nach ([13], Theorem 4.6) gilt folgende Abschätzung für den hierbei gemachten Fehler:

**Satz 1.6.1:** *Seien $M, S \subset \mathbb{R}^2$ wie oben bestimmte Mengen.*
*Dann gilt für Punkt $p \in \mathbb{R}^2$, der nicht innerhalb der Menge $co(S)$ liegt:*

$$dist(p, co(S)) \leq \frac{xmax - xmin}{k}.$$

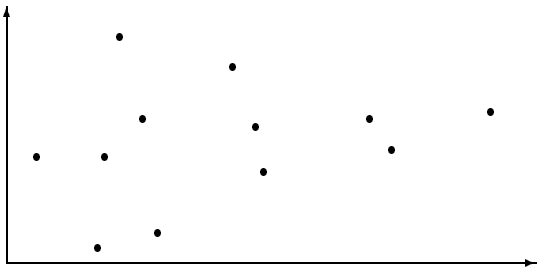
*Hierbei bezeichnet*

- *$xmax$ die maximale x-Koordinate der Punkte $p_1, \ldots, p_N$,*
- *$xmin$ die kleinste x-Koordinate der Punkte $p_1, \ldots, p_N$,*
- *$k$ die Anzahl der Streifen.*

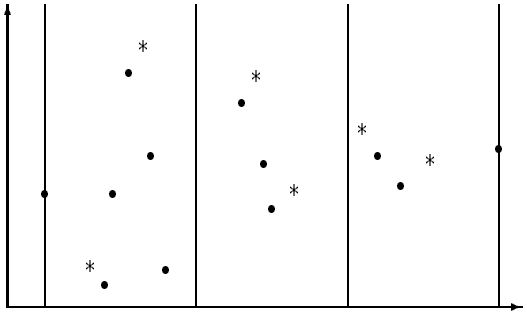
$\diamond$

Mit dem nachfolgenden Beispiel wird diese Tatsache unterstrichen sowie das Ablaufschema des Algorithmus veranschaulicht.

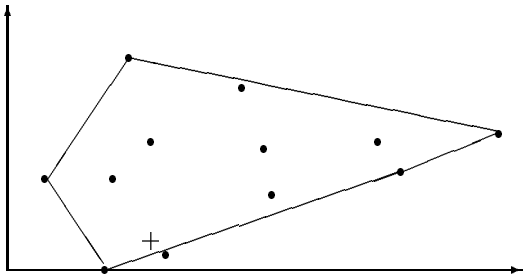
**Beispiel 1.6.2:** Betrachte folgende Punktemenge in der Ebene:



Nach der Unterteilung in $k = 3$ Streifen, können wir in jedem Streifen die extremalen Punkte bestimmen. Diese sind mit (*) gekennzeichnet. Es ergibt sich folgende Situation:



Als konvexe Menge erhalten wir:



Offensichtlich gehört der Punkt, der mit (+) gekennzeichnet ist, zu $co(M)$ ist jedoch nicht in $co(S)$ enthalten. $\diamond$

Da wir zur Durchführung der Schritte (1)-(5) maximal einen Aufwand von O(N) benötigen, und da die konvexe Hülle mit einem Aufwand von $O(k)$ bestimmt werden kann, weil Punkte sortiert nach ihrem x-Wert vorliegen, erhalten wir folgende Abschätzung für die Gesamtlaufzeit:

**Satz 1.6.3:** *Zur Bestimmung einer approximierenden Menge für die konvexe Hülle von $N$ Punkten aus der Ebene benötigen wir bei $k$ Streifen einen Aufwand von $O(N+k)$.* ◇

# Kapitel 2

# Implementierung der Grahamschen Abtastung

Bei der Umsetzung des Grahamschen Algorithmus (vgl. 1.2) müssen wir nachfolgende Teilprobleme lösen:

1. Einlesen der Daten in eine geeignete Struktur.
2. Bestimmung eines inneren Punktes einer konvexen Menge.
3. Transformation einer Punktemenge.
4. Sortierung der Punktemenge aufsteigend nach den Sortierkriterien. polarer Winkel und Entfernung zum Ursprung.
5. Ablegen der Punktinformationen in einer verketteten Liste.
6. Durchführung der Grahamschen Abtastung, um innere Punkte zu bestimmen.

Die im folgenden beschriebenen Funktionen befinden sich in dem C-Programm *graha1.c*. Eine genauere Auflistung der benötigten Parameter sowie deren Bedeutung kann der Anwender im Dokumentationsfile *graham.dvi* oder im angegebenen C-File nachlesen.

## 2.1 Die Funktion *lies_info*

Wir gehen jetzt davon aus, daß die x- und y-Koordinaten einer Punktemenge $p_1, \ldots, p_s \in \mathbb{R}^2$ zeilenweise in einem File hinterlegt sind. Diese Koordinaten werden zunächst in Elementen vom Typ *point* abgespeichert. Hierbei handelt es sich um eine Struktur mit den folgenden Komponenten:

```
struct point {
              double x;
              double y;
            }
```

Um die Punktemenge wie gefordert sortieren zu können, und um die Punkte in einer verketteten Liste abspeichern zu können, wird diese Information innerhalb der Funktion *graham_abtastung*, die den Programmablauf steuert, in Elementen der Struktur *DATES* abgelegt. Diese ist folgendermaßen definiert:

```
struct DATES {
                double x_pos;
                double y_pos;
                double range;
                double angle;
                int is_edge;
              }
```

Die angegebenen Strukturen sind in dem Headerfile *graham.h* definiert.

## 2.2 Die Funktion *best_in_punkt*

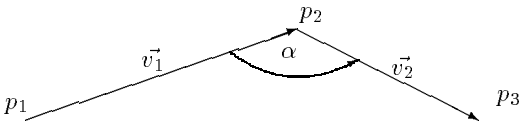
Wollen wir einen inneren Punkt der Menge $co(\{p_1, \ldots, p_s\})$ bestimmen, so ist dies nur möglich, falls drei Punkte $p_{i_1}, p_{i_2}$ und $p_{i_3}$ $(1 \leq i_1 < i_2 < i_3 \leq s)$ existieren, die nicht auf einer Geraden liegen. Mit Hilfe des nachfolgenden Satzes sind wir in der Lage entscheiden zu können, ob drei Punkte auf einer Geraden liegen oder nicht.

**Satz 2.2.1:** *Seien* $p_1 = (x_1, y_1), p_2 = (x_2, y_2)$ *und* $p_3 = (x_3, y_3) \in \mathbb{R}^2$. *So gilt:*

$$\det \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \begin{array}{ll} < & p_1, p_2 \text{ und } p_3 \text{ liegen rechtsgekrümmt} \\ = 0 & p_1, p_2 \text{ und } p_3 \text{ liegen auf einer Geraden} \\ > & p_1, p_2 \text{ und } p_3 \text{ liegen linksgekrümmt.} \end{array}$$

◇

*Beweis.* Betrachte folgende Situation:



Wir ermitteln einen Homomorphismus $F$ so, daß die Einheitsvektoren $e_1, e_2$ auf $\vec{v_1}, \vec{v_2}$ abgebildet werden. Dieser Homomorphismus ist eindeutig. (vgl. [12]; §9, Satz 2). $F$ muß offensichtlich wie folgt gewählt werden:

$$F : \mathbb{R}^2 \to \mathbb{R}^2 \qquad x \mapsto (v_1 v_2)x$$

wobei

$$v_1 = \begin{pmatrix} x_2 - x_1 \\ y_2 - y_1 \end{pmatrix} \text{ und } v_2 = \begin{pmatrix} x_3 - x_2 \\ y_3 - y_2 \end{pmatrix}.$$

Bestimme nun $\det F$:

$$\det F = \det(v_1 v_2) = \det \begin{vmatrix} (x_2 - x_1) & (x_3 - x_2) \\ (y_2 - y_1) & (y_3 - y_2) \end{vmatrix} =$$

$$= x_2 y_3 - x_2 y_2 - x_1 y_3 + x_1 y_2 - x_3 y_2 + x_2 y_2 + x_3 y_1 - x_2 y_1$$

$$= \det \begin{vmatrix} x_2 & y_2 \\ x_3 & y_3 \end{vmatrix} - \det \begin{vmatrix} x_1 & y_1 \\ x_3 & y_3 \end{vmatrix} + \det \begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix}$$

$$= \det \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \qquad (*)$$

Mit der Abbildung

$$\tau : \mathbb{R}^2 \to \mathbb{R}^2 \qquad \begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} y \\ -x \end{pmatrix}$$

erhalten wir

$$\begin{aligned} \det F = & \quad \det(v_1 v_2) = v_1^T \tau(v_2) \\ = & \quad \|v_1\|_2 \|\tau(v_2)\|_2 \cos(\underbrace{\angle(v_1, \tau(v_2))}_{=:\beta}) = \|v_1\|_2 \|v_2\|_2 \cos\beta. \end{aligned}$$

Zwischen den beiden Winkeln $\alpha$ und $\beta$ gelten nachfolgende Beziehungen, die mittels Fallunterscheidung zu verifizieren sind:

$$\begin{aligned} \alpha \in ]0, \pi[ \quad &\Leftrightarrow \quad \beta \in ]\frac{\pi}{2}, \pi[, \\ \alpha \in ]\pi, 2\pi[ \quad &\Leftrightarrow \quad \beta \in ]0, \frac{\pi}{2}[. \end{aligned}$$

Damit folgt:

- $\det F < 0 \Leftrightarrow \cos\beta < 0 \Leftrightarrow \beta \in ]\frac{\pi}{2}, \pi[$
  $\Leftrightarrow \alpha \in ]0, \pi[ \Leftrightarrow p_1, p_2, p_3$ sind rechtgekrümmt angeordnet,

- $\det F > 0 \Leftrightarrow \cos\beta > 0 \Leftrightarrow \beta \in ]0, \frac{\pi}{2}[$
  $\Leftrightarrow \alpha \in ]\pi, 2\pi[ \Leftrightarrow p_1, p_2, p_3$ liegen linksgekrümmt,
- $\det F = 0 \Leftrightarrow \cos\beta = 0 \Leftrightarrow \beta = \frac{\pi}{2}, \frac{3}{2}\pi$
  $\Leftrightarrow \alpha = 0, \pi \Leftrightarrow p_1, p_2$ und $p_3$ liegen auf einer Geraden.

Unter Berücksichtigung der Aussage (*) folgt auf diese Weise die Behauptung. □

Haben wir drei Punkte $p_{i_1}, p_{i_2}$ und $p_{i_3}$ gefunden, die nicht auf einer Geraden liegen, so ist der Schwerpunkt des Dreiecks $p_{i_1}p_{i_2}p_{i_3}$ ein innerer Punkt der Menge $co(\{p_1, \ldots, p_s\})$. Numerisch bestimmen wir den Punkt $p$ nach der Formel

$$p = \frac{1}{3}(p_{i_1} + p_{i_2} + p_{i_3}).$$

## 2.3 Die Funktion *trafo_punkte*

Mit Hilfe dieser Funktion wird eine gegebenen Punktemenge $p_1, \ldots, p_s \in \mathbb{R}^2$ so transformiert, daß der Punkt $p$ aus 2.2 im Ursprung des Koordinatensystems zu liegen kommt. Um dies zu erreichen, subtrahieren wir von den x-Koordinaten der Punkte $p_1, \ldots, p_s$ die x-Koordinate des Punktes $p$. Analog verfahren wir mit den y-Koordinaten.

## 2.4 Die Funktion *vor_sort*

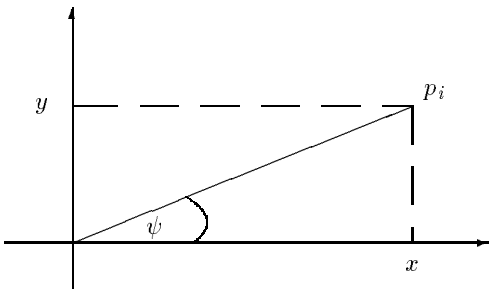



Abbildung 2.1: Veranschaulichung der Größen nach denen die Punktemenge sortiert wird

Bevor die Sortierung nach den Kriterien polarer Winkel $\psi$ und Entfernung zum Ursprung $d$ (siehe Abbildung 2.1) durchgeführt werden kann, müssen diese

Größen in der Prozedur *vor_sort* zunächst für jeden Punkt berechnet werden. Die Ergebnisse werden in Elementen vom Typ *DATES* in den Komponenten *angle* und *range* gespeichert. Anschließend werden die Punkte mittels der C-Standardfunktion *qsort* sortiert. Die Grundlage dieser Routine bildet der Quicksortalgorithmus (vgl. 1.3).

## 2.5 Die Funktion *gen_liste*

Bevor die Grahamsche Abtastung durchgeführt werden kann, müssen die Punkte in einer verketteten Liste gespeichert werden. Diese Struktur ist wie folgt definiert:

**Definition 2.5.1:**
Eine *doppelt verkettete Liste* ist eine Folge von Knoten mit folgender Struktur:

| DATA | |
|---|---|
| LINKPRED | LINKSUCC |

In dem Feld *DATA* werden Daten gleichen Typs abgespeichert. LINKSUCC und LINKPRED sind Kettenfelder. Ein Kettenfeld enthält einen Zeiger auf einen anderen Knoten. Die Kettenfelder LINKSUCC weisen jeweils auf den nachfolgenden Knoten und die Felder LINKPRED auf den Vorgänger. Der Vorgänger des Knotens mit Nummer 1 ist der Knoten mit Nummer N. Der Nachfolger des Knotens mit Nummer N ist der Knoten mit Nummer 1. ◇

Wir benutzen Elemente vom Typ *DATES* zur Realisierung einer doppelt verketteten Liste. Die Komponenten *angle* und *range* werden nach dem Sortieren der Punkte nicht mehr benötigt. Wir können sie also verwenden, um Kettenfelder zu verwirklichen. In der Komponente *angle* wird die Nummer des Vorgängerknotens und in *range* die Nummer seines Nachfolgers gespeichert.

**Bemerkung:** Statt der benutzten doppelt verketteten Liste hätten wir die Information auch in gewöhnlichen Feldern speichern können. Dies wäre einfacher zu realisieren gewesen. Bei Verwendung verketteter Listen sind jedoch Löschoperationen, wie wir sie in der Grahamschen Abtastung durchführen müssen, um innere Punkte zu eliminieren, wesentlich schneller ausführbar.

## 2.6 Die Funktion *get_one_edge*

Um die Grahamsche Abtastung starten zu können, müssen wir zunächst einen Punkt aus der transformierten Punktemenge bestimmen, der ein Eckpunkt der

konvexen Hülle ist. Dazu wird der Punkt $p$ mit der kleinsten y-Koordinate ermittelt. Gibt es mehrere Punkte mit dieser Eigenschaft, so wählen wir unter diesen den Punkt mit der größten x-Koordinate.

Dies ist notwendig, um zum einen einen Einstieg für die Abtastung zu bekommen, und zum anderen, um auf einfache Weise ein Abbruchkriterium für das Durchlaufen der doppelt verketteten Liste zu finden. Das Abbruchkriterium lautet unter diesen Voraussetzungen: Wir betrachten solange drei aufeinanderfolgende Punkte $p_i, p_{i+1}, p_{i+2}$ aus der verketteten Liste, die noch nicht als innere Punkte erkannt wurden, und wenden Kriterium K3 an, um zu entscheiden, ob der Punkt $p_{i+1}$ ein innerer Punkt ist, bis gilt: $p_{i+1} = p$. Diese Situation darf nicht durch "Backtracking" entstanden sein, d.h. beim vorherigen Vergleich von drei Punkten ist festgestellt worden, daß diese nicht linksgekrümmt angeordnet waren.

## 2.7 Die Funktion *graham_scan*

Mit dieser Funktion werden mit Hilfe von Kriterium K3 alle Punkte bestimmt, die keine Eckpunkte der konvexen Hülle sind. Diese Punkte werden im Laufe der Abtastung markiert, so daß am Ende alle unmarkierten Punkte alle Eckpunkte der konvexen Hülle darstellen. Nach dieser Markierung und nachdem die Transformation aus 2.3 rückgängig gemacht wurde, werden die Koordinaten aller Eckpunkte in einem Feld mit Elementen vom Typ *point* gespeichert.

**Bemerkung:**
Der Programmablauf wird mittels der Funktion *graham_abtastung* gesteuert. Möchte der Anwender hierzu Detailinformation (z.B. Beschreibung der Parameter), so findet er diese sowohl im Quellcode des C-Programms *graha1.c* als auch in der Dokumentation *graham.dvi*.

# Kapitel 3

# Anwendung der Algorithmen in der Ebene

Wie in Kapitel 0 beschrieben, benötigen wir Algorithmen zur Bestimmung konvexer Hüllen unter anderem zur Approximation mengenwertiger Integrale mit Hilfe von Quadraturformeln. Um die Güte eines Approximationsverfahrens testen zu können, das auf der Treppen-, Trapez- und Simpsonregel beruht, gehen wir wie folgt vor:

Zunächst muß eine Referenzlösung für das gesuchte mengenwertige Integral bestimmen werden. In der Regel kennen wir jedoch keine exakte Lösung für dieses Integral, so daß nur eine "gute" Näherung zum Vergleich herangezogen werden kann. Diese erhalten wir, indem wir ein Approximationsverfahren mit hoher Konvergenzordnung auswählen und dieses mit einem großen Diskretisierungsparameter ($N \geq 1000$) durchrechnen. Da bei diesem Verfahren gewichtete konvexe Hüllen addiert werden, erhalten wir eine konvexe Menge $Q = co(q_1, \ldots, q_s)$, die im folgenden als Vergleichsmaßstab dient.

Es stellt sich uns nun die Frage, wie wir ein Maß für den anfallenden Fehler erhalten. Dies wird nachfolgend beschrieben.

## 3.1 Fehlerabschätzung mittels Hausdorffabstand

Berechnen wir eine Lösung $\tilde{y}$ für ein Riemannintegral $\int_a^b f(t)dt =: y$ mit Hilfe eines Approximationsverfahrens, so ist der absolute Fehler $w = |y - \tilde{y}|$ ein Maß für die Güte der Approximation. Für den mengenwertigen Fall muß der Abstandsbegriff erweitert werden.

**Definition 3.1.1:**
Gegeben seien zwei nichtleere Mengen $A, B \in \mathbb{R}^n$ und ein Punkt $x \in \mathbb{R}^n$. Dann heißt

$$dist(x, A) := \inf_{a \in A} \|x - a\|_2$$

*Abstand* des Punktes $x$ von der Menge $A$ und $d(\cdot, A)$ *Distanzfunktion* der Menge $A$. Als *einseitigen Hausdorffabstand* der Mengen $A$ und $B$ definiert man

$$d(A, B) := \sup_{a \in A} dist(a, B).$$

Als *Hausdorffabstand* von $A$ und $B$ bezeichnen wir

$$haus(A, B) := \max(dist(A, B), dist(B, A)).$$

Sei $A \in \mathbb{R}^n$ eine Menge und $< \cdot, \cdot >$ das euklidische Skalarprodukt im $\mathbb{R}^n$. Dann heißt

$$\delta^*(\cdot, A): \quad \mathbb{R}^n \to \mathbb{R} \qquad l \to \sup_{a \in A} < l, a >$$

*Stützfunktion der Menge* $A$. Ist $A = \emptyset$ so setzt man

$$\delta^*(\cdot, \emptyset) = -\infty.$$

Sei $A \subset \mathbb{R}^n$ eine nichtleere Teilmenge und $l \in \mathbb{R}^n$.
$a^0 \in A$ heißt *Stützpunkt*, wenn es ein stetiges, lineares Funktional $f : \mathbb{R}^n \to \mathbb{R}$ gibt mit $f(\cdot) \neq 0$ und $\sup_{a \in A} f(a) = f(a^0)$. Die Menge aller Stützpunkte bzgl. des Funktionals $f(\cdot)$ bezeichnen wir mit $Y(f(\cdot), A)$, während wir die Menge aller Stützpunkte bzgl. des Funktionals $f_l(\cdot) =< l, \cdot >$ mit $Y(l, A)$ benennen, da wir $l$ mit $f_l(\cdot)$ identifizieren können. $y(l, A)$ ist ein Stützpunkt aus der Menge $Y(l, A)$. $\diamond$

Ein Maß für den Fehler erhalten wir auf die folgende Weise:

**Satz 3.1.2:** *Sei* $P^M := \{l^k | k = 0, \ldots, M-1\} \subset \mathbb{R}^2$, $(M \in \mathbb{N})$ *die endliche Menge der Richtungen der Form*

$$l^k = \begin{pmatrix} cos(\alpha_k) \\ sin(\alpha_k) \end{pmatrix} \in \partial B_1(0);\ \alpha_k = \frac{2\pi k}{M}\ (k = 0, \ldots, M-1).$$

*Sei* $P = co(p_1, \ldots, p_r)$ *die konvexe Hülle der Punkte* $p_1, \ldots, p_r$ *und* $Q = co(q_1, \ldots, q_s) \in \mathbb{R}^2$. *Für den Hausdorffabstand der Mengen* $P$ *und* $Q$ *gilt nun in der Ebene:*

$$haus(P, Q) \quad = \quad \sup_{l \in B_1(0)} |\delta^*(l, P) - \delta^*(l, Q)| = \tag{3.1}$$

$$= \sup_{l \in \partial B_1(0)} |\delta^*(l,P) - \delta^*(l,Q)| = \tag{3.2}$$

$$= \sup_{l \in \partial B_1(0)} |\max_{i=1,\dots,r} < l, p_i > - \max_{i=1,\dots,s} < l, q_i > | \approx \tag{3.3}$$

$$\approx \max_{j=1,\dots,M} |\max_{i=1,\dots,r} < l^j, p_i > - \max_{i=1,\dots,s} < l^j, q_i > |. \tag{3.4}$$

$\diamond$

*Beweis.* Einen Beweis für die Gleichung 3.1 findet man für nichtleere, kompakte, konvexe Mengen des $\mathbb{R}^n$ in [5].
Wegen

$$\delta^*(0_{\mathbb{R}^n}), P) - \delta^*(0_{\mathbb{R}^n}, Q) = 0$$

und der positiven Homogenität

$$\begin{aligned} |\delta^*(l,P) - \delta^*(l,Q)| &= \|l\|_2 |\frac{1}{\|l\|_2}(\delta^*(l,P) - \delta^*(l,Q))| \\ &\leq |\delta^*(\frac{1}{\|l\|_2}l, P) - \delta^*(\frac{1}{\|l\|_2}l, Q)| \end{aligned}$$

für $l \in B_1(0) \setminus \{0_{\mathbb{R}^n}\}$ folgt die Gleichung 3.2. Die dritte Gleichung resultiert aus der Definition der Stützfunktion und der Tatsache, daß die Menge $P$ und $Q$ beschränkte Teilmengen des $\mathbb{R}^n$ sind. Da in 3.4 im Gegensatz zu 3.3 nur noch endlich viele Richtungen betrachtet werden, ergibt sich die angegebene Approximation. $\square$

**Bemerkung:** Die Aussagen 3.1-3.3 gelten für beliebige nichtleere, konvexe und kompakte Teilmengen des $\mathbb{R}^n$. Lediglich die Näherung 3.4 gilt für Mengen aus der Ebene.

Wir wollen nun untersuchen, wie groß der Fehler ist, der bei dieser Approximation gemacht wird. Eine Antwort hierauf liefert der nachfolgende Satz:

**Satz 3.1.3:** *Sei* $P^M := \{l^k | k = 0, \dots, M-1\} \subset \mathbb{R}^2$ *die endliche Menge der Richtungen der Form*

$$l^k = \begin{pmatrix} cos(\alpha_k) \\ sin(\alpha_k) \end{pmatrix} \in \partial B_1(0);\ \alpha_k = \frac{2\pi k}{M}\ (k = 0, \dots, M-1)$$

*und* $P = co(p_1, \dots, p_r)$ *sowie* $Q = co(q_1, \dots, q_s)$.
*Für* $M \geq 7$ *erfüllt diese Menge die Bedingungen*

1. *Für jedes* $l \in \mathbb{R}^2$ *existieren* $\alpha_i(l) \geq 0$ *(i=1,...,M) mit*

$$l = \sum_{i=1}^{M} \alpha_i(l) l^i$$

2. *Für* $\epsilon := \frac{2\pi}{M} \in ]0,1[$ *sind für beliebiges* $l \in \mathbb{R}^2$ *alle Richtungen* $l^i$ *mit positiven Koeffizienten* $\alpha_i(l)$ *benachbart, d.h. falls* $\alpha_i(l)\alpha_j(l) > 0$ *gilt:*

$$\|l^i - l^j\|_2 \leq \epsilon.$$

*Daher gilt für die obige Approximation:*

$$|haus(P,Q) - \max_{j=1,\ldots,M} | \max_{i=1,\ldots,r} < l^j, p_i > - \max_{i=1,\ldots,s} < l^j, q_i > || \leq \frac{\|P\| + \|Q\|}{1-\epsilon}\epsilon$$

*wobei*

$$\|P\| := \sup_{p \in P} \|p\|_2.$$

◇

*Beweis.* Dieser Satz ergibt sich unmittelbar aus den Sätzen [3, Satz 3.1.4.5] und [3, Satz 3.1.4.6]. □

Die Funktion *haus_abstand*, die der Anwender im C-File *aumann.c* findet, berechnet wie in 3.4 eine Näherung für den Hausdorffabstand zweier Polytope. Es ist hierbei zu beachten, daß der Parameter $M$ wesentlich größer als die Zahl der Eckpunkte der betrachteten Polytope sein muß, da sonst nicht genügend Richtungen betrachtet werden, und die Abschätzung zu grob wird. Wir verwenden für diesen Parameter den Wert M=2000.

## 3.2 Eine Schätzung für die Konvergenzordnung

Wie bereits in Kapitel 0 angedeutet spielt bei der Charakterisierung numerischer Verfahren die Konvergenzordnung eine wichtige Rolle.

**Definition 3.2.1:**
Sei $B = \int_a^b F(t)dt$ ein Aumannintegral und $w(h)$ die Näherungslösung, die mittels einer Quadraturformel bei gegebener Schrittweite $h = \frac{b-a}{N}$ berechnet worden ist. Das Approximationsverfahren ist *konvergent von der Ordnung* $p$, falls es ein $C > 0$ gibt, mit:

$$haus(w(h), B) \leq C \cdot h^p.$$

Ein maximales $p \in \mathbb{N}$ mit dieser Eigenschaft bestimmt die Konvergenzordnung $ord_B$ des Verfahrens beim Integral $B$. ◇

Wollen wir die Konvergenzordnung eines Verfahrens bestimmen, so ergibt sich ein Problem: Die exakte Lösung des Aumannintegrals ist häufig nicht bekannt.

Daher müssen wir nach einer Möglichkeit suchen, diese Ordnung ohne diese Information abzuschätzen. Dazu gehen wir folgendermaßen vor: Sei $Q^V_{n_k}$ die Näherungslösung, die mit Hilfe eines Verfahrens $V$ mit einem Diskretisierungsparameter $n_k$ ($k = 1, 2; n_1 \neq n_2; n_1, n_2 \in \mathbb{N}$) bestimmt wurde. Wir definieren

$$w^V_k := haus(Q^V_{n_k}, \int_a^b F(t)dt).$$

Gesucht ist nun ein maximales $p$ mit

$$w^V_k \leq c \cdot (\frac{b-a}{n_k})^p.$$

Wir nehmen nun an, daß diese Ungleichung mit Gleichheit erfüllt ist. Dann folgt:

$$w^V_k = c \cdot (\frac{b-a}{n_k})^p.$$

Somit folgt, da $c \neq 0$:

$$\frac{w^V_1}{w^V_2} = \frac{c \cdot (\frac{b-a}{n_1})^p}{c \cdot (\frac{b-a}{n_2})^p} = \frac{n_2^p}{n_1^p} = \left(\frac{n_2}{n_1}\right)^p.$$

Für $p$ ergibt sich hieraus:

$$p = \frac{\ln \frac{w^V_1}{w^V_2}}{\ln \frac{n_1}{n_2}}$$

Zu gegebenen $n_1$ und $n_2$ müssen die Hausdorffabstände $w^V_1$ und $w^V_2$ berechnet werden und wir erhalten eine Schätzung für die Konvergenzordnung des Verfahrens $V$. Soll zu einer Folge von Diskretisierungsparametern $(N_1, \ldots, N_s)$ ($s \in \mathbb{N}$) die jeweilige Konvergenzordnung abgeschätzt werden, so haben wir zwei Möglichkeiten: Zum einen können wir für $k = 1, \ldots, s-1$ $n_1 = N_k$ und $n_2 = N_{k+1}$ setzen und die Konvergenzordnung jeweils nach obiger Formel berechnen, oder wir definieren $n_1 = N_j$ und $n_2 = N_k$ wobei $j \in \{1, \ldots, s\}$ fest gewählt ist. Bei unseren Untersuchungen machen wir von der zweiten Möglichkeit Gebrauch und setzen $j = 2$, da die erste Näherung unter Umständen eine erhebliche Abweichung von der Lösung aufweisen könnte.

## 3.3 Eine obere Schranke für die Laufzeit

Wesentlich für die Güte eines Algorithmus ist seine benötigte Laufzeit. Um die Laufzeit eines auf einer Quadraturformel beruhenden Approximationsverfahrens nach oben abschätzen zu können, ist von entscheidender Bedeutung, daß wir die Anzahl der Eckpunkte der zwischenzeitlich auftretenden Mengen in den Griff bekommen. Dies leistet für spezielle Aumannintegrale der nachfolgender Satz:

**Satz 3.3.1:** *Gegeben sei ein Aumannintegral der Form*

$$\int_a^b F(t)dt = \int_a^b \begin{pmatrix} x_1(t) \\ x_2(t) \end{pmatrix} [c,d]dt$$

*mit* $a \neq b$, $c \neq d$, $c < d$ *und* $x_i : [a,b] \to \mathbb{R}$ $(i = 1,2)$ *reelle Funktionen.*
*Sei* $A_N$ *die approximierende Menge, die mit Hilfe einer Summenformel*

$$\sum_{i=0}^{N} a_i co(F(t_i))$$

*erzeugt wurde.*
*Dabei seien* $a_i \geq 0$; $i = 0,\ldots,N$ *Koeffizienten,* $F : [a,b] \to \mathcal{P}(\mathbb{R}^n)$ *eine mengenwertige Abbildung und* $t_i := a + \frac{b-a}{N} \cdot i$; $i = 0,\ldots,N$ *eine äquidistante Unterteilung des Intervalls* $[a,b]$. *Sind bei der Anwendung einer Summenformel* $M \leq N+1$ *Koeffizienten* $a_i \neq 0$, *so gilt für die Eckpunktzahl* $anz(A_N)$ *der Menge* $A_N$*:*

$$anz(A_N) \leq 2 \cdot M.$$

◇

*Beweis.* Sei $N \in \mathbb{N}$ beliebig gewählt. Wir führen eine Induktion über $M$ durch:
$\underline{M = 1}$:
Wir erhalten als approximierende Menge

$$A_N = a_{i_0} \cdot co(F(t_{i_0})) = a_{i_0} \left[ c \begin{pmatrix} x_1(t_{i_0}) \\ x_2(t_{i_0}) \end{pmatrix}, d \begin{pmatrix} x_1(t_{i_0}) \\ x_2(t_{i_0}) \end{pmatrix} \right]^*$$

wobei $a_{i_0} \neq 0$. Dies ist eine Strecke. Daher besitzt die Menge $A_N$ höchstens zwei Eckpunkte.
$\underline{M \to M+1}$:
Sei $A_M := \sum_{j=0}^{M-1} a_{i_j} co(F(t_{i_j}))$ die Zwischenmenge, die sich durch Addition von $M$ Strecken ergeben hat. Diese besitzt nach Voraussetzung höchstens $2M$ Eckpunkte. Wir zeigen nun, daß für die Menge $A_{M+1} = A_M + a_{i_M} co(F(t_{i_M}))$, wobei $i_M \neq i_j \ \forall j \in \{0,\ldots,M-1\}$, gilt:

$$anz(A_{M+1}) \leq 2(M+1). \tag{3.5}$$

Es sind zwei Fälle zu unterscheiden:
a) $\underline{\nexists d_2 \in int(A_M)}$:
In dieser Situation gilt folglich:

$$dim\ A_M \in \{0,1\}.$$

$A_M$ ist also ein Punkt oder eine Strecke, d.h. die Aussage 3.5 ist trivialerweise erfüllt.

b) $\exists d_2 \in int(A_M)$:

Die Punkte $\{q_1, \ldots, q_{2M}\}$ der Menge $A_M$ werden wie folgt durchnummeriert: Wir bestimmen den Punkt $d_1$ mit der kleinsten y-Koordinate. Gibt es zwei Punkte mit dieser Eigenschaft, so wählen wir den Punkt mit der kleineren x-Koordinate. Jetzt sortieren wir die Punkte aufsteigend nach dem orientierten Zwischenwinkel gegen den Uhrzeigersinn $\alpha_i = \angle(d_2 d_1, d_2 q_i)$ $(i = 1 \ldots, 2M)$.
Bezeichnen wir mit $P$ die Menge $a_{i_M} co(F(t_{i_M}))$, so sind zwei Fälle zu unterscheiden:

Fall 1: Die Menge $P$ besteht nur aus einem Punkt $p$. Dann folgt:

$$A_M + P = co(\{q_i + p \mid i = 1, \ldots, 2M\})$$

besitzt höchstens $2M$ Eckpunkte. Somit folgt:

$$anz(A_{M+1}) = anz(A_M + P) \leq 2M < 2M + 2 = 2(M + 1).$$

Fall 2: Die Menge $P$ besitzt zwei Eckpunkte $p_1$ und $p_2$, wobei o.E. $p_2^x \geq p_1^x$ sei. Diese Einschränkung ist erforderlich, da bei den nachfolgenden Aussagen die Lage der Punkte im Raum von entscheidender Bedeutung ist. Gilt $p_2^x < p_1^x$, so ersetze im folgenden $p_1$ durch $p_2$ und umgekehrt.

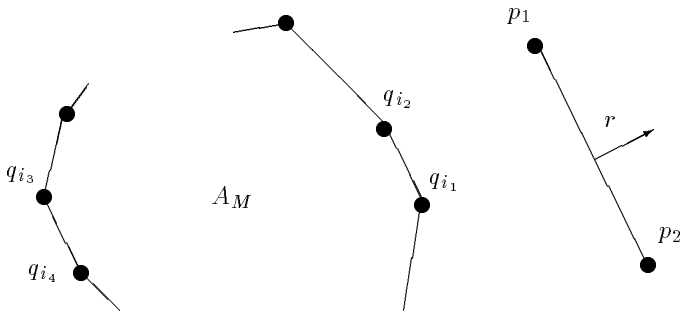
$$\Rightarrow \exists r \in \mathbb{R}^2 \; mit \; \|r\|_2 = 1 \; wobei \; r = \begin{pmatrix} \cos\alpha \\ \sin\alpha \end{pmatrix} (\alpha \in [0, \pi]) \; und \; r \perp P.$$

Es sind nun vier Fälle zu unterscheiden:

Fall 2.1: $\exists i_1 < i_2 < i_3 < i_4$ mit $i_k \in \{1, \ldots, 2M\}, k = 1, \ldots, 4$

$$mit \; q_{i_1} = y(r, A_M), \; q_{i_2} = y(r, A_M), \; q_{i_3} = y(-r, A_M), \; q_{i_4} = y(-r, A_M)$$

Wir betrachten die Situation:




Nun gilt:

$$\bigvee_{i_3 > i > i_2} q_i + p_2 \in int(A_M + P) \tag{3.6}$$

$$\bigvee_{i_4<i,i<i_1} q_i + p_1 \in int(A_M + P) \tag{3.7}$$

$$q_{i_2} + p_2 \in co(\{q_{i_2} + p_1, q_{i_1} + p_2\}) \tag{3.8}$$

$$q_{i_1} + p_1 \in co(\{q_{i_2} + p_1, q_{i_1} + p_2\}) \tag{3.9}$$

$$q_{i_3} + p_2 \in co(\{q_{i_3} + p_1, q_{i_4} + p_2\}) \tag{3.10}$$

$$q_{i_4} + p_1 \in co(\{q_{i_3} + p_1, q_{i_4} + p_2\}). \tag{3.11}$$

Zunächst zeigen wir die Richtigkeit der Aussage 3.8. Die Behauptungen 3.9, 3.10 und 3.11 lassen sich auf ähnliche Weise zeigen:

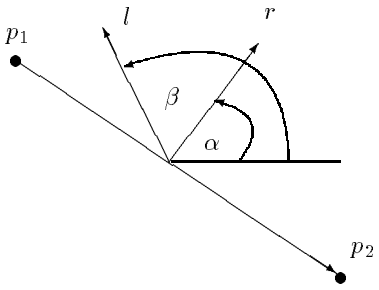
$$q_{i_2} + p_2 \in co(q_{i_2} + p_1, q_{i_1} + p_2)$$

$$\Leftrightarrow \bigvee_{l \in \partial B_1(0)} < l, q_{i_2} + p_2 > \leq \delta^*(l, co(\{q_{i_2} + p_1, q_{i_1} + p_2\}))$$

$$\Leftrightarrow \bigvee_{l \in \partial B_1(0)} < l, q_{i_2} + p_2 > \leq \max(< l, q_{i_2} + p_1 >, < l, q_{i_1} + p_2 >).$$

Wir müssen jetzt zwei Fälle unterscheiden:

a) Sei $l = \begin{pmatrix} \cos\beta \\ \sin\beta \end{pmatrix}$ mit $\alpha \leq \beta \leq \alpha + \pi$.
Somit ergibt sich folgende Situation:




Es gilt nun:

$$< l, p_2 > \leq < l, p_1 > . \tag{3.12}$$

Diese Aussage erhalten wir wie folgt:

$$< l, p_2 > \leq < l, p_1 >$$

$$\Leftrightarrow < l, p_2 - p_1 > \leq 0$$

$$\Leftrightarrow \ \|l\|_2 \|p_2 - p_1\|_2 \cdot \cos(\angle(l, p_2 - p_1)) \leq 0$$

$$\Leftrightarrow \ \cos(\angle(l, p_2 - p_1)) \leq 0$$

$$\Leftrightarrow \frac{\pi}{2} + \beta - \alpha \in [\frac{\pi}{2}, \frac{3\pi}{2}]$$
$$\Leftrightarrow \beta \in [\alpha, \alpha + \pi].$$

Dies gilt nach Voraussetzung, also folgt die Aussage 3.12.
Damit resultiert die nachfolgende Abschätzung:

$$\begin{aligned} < l, q_{i_2} + p_2 > = & < l, q_{i_2} > + < l, p_2 > \leq < l, q_{i_2} > + < l, p_1 > \\ = & < l, q_{i_2} + p_1 > \leq \max(< l, q_{i_2} + p_1 >, < l, q_{i_1} + p_2 >). \end{aligned}$$

b) Sei $l = \begin{pmatrix} \cos\beta \\ \sin\beta \end{pmatrix}$ mit $\beta \notin [\alpha, \alpha + \pi]$. Analog zu 3.12 läßt sich zeigen:

$$< l, q_{i_2} > < < l, q_{i_1} > .$$

Damit erhalten wir die folgende Abschätzung:

$$\begin{aligned} < l, q_{i_2} + p_2 > = & < l, q_{i_2} > + < l, p_2 > < < l, q_{i_1} > + < l, p_2 > \\ = & < l, q_{i_1} + p_2 > \leq \max(< l, q_{i_2} + p_1 >, < l, q_{i_1} + p_2 >). \end{aligned}$$

Somit ist die Behauptung 3.8 gezeigt.

Wir zeigen nachfolgend die Aussage 3.6 und weisen darauf hin, daß sich 3.7 analog zeigen läßt:

$$\bigvee_{i_3 > i > i_2} q_i + p_2 \in int(A_M + P)$$
$$\Leftrightarrow \bigvee_{l \in \partial B_1(0)} \bigvee_{i_3 > i > i_2} < l, q_i + p_2 > < \delta^*(l, A_M) + \delta^*(l, P)$$
$$\Leftrightarrow \bigvee_{l \in \partial B_1(0)} \bigvee_{i_3 > i > i_2} < l, q_i + p_2 > < \max_{j=1,\ldots,2M} < l, q_j > + \max_{j=1,2} < l, p_j > .$$

Wir betrachten nun die Negation dieser Aussage:

$$\bigwedge_{l \in \partial B_1(0)} \bigwedge_{i_2 < i < i_3} < l, q_i > + < l, p_2 > \geq \max_{j=1,\ldots,2M} < l, q_j > + \max_{j=1,2} < l, p_j > \quad (3.13)$$

Da

$$< l, q_i > \leq \max_{j=1,\ldots,2M} < l, q_j >$$

und

$$< l, p_i > \leq \max_{j=1,2} < l, p_j >$$

muß gelten:

$$\bigwedge_{l \in \partial B_1(0)} \bigwedge_{i_2 < i < i_3} < l, q_i > + < l, p_2 > = \max_{j=1,\ldots,2M} < l, q_j > + \max_{j=1,2} < l, p_j > .$$

$\Rightarrow \exists l \in \partial B_1(0)$ und $\exists i$ mit $i_2 < i < i_3$ mit folgenden Eigenschaften:

$$l = \begin{pmatrix} \cos\beta \\ \sin\beta \end{pmatrix} \beta \in [0, 2\pi] \tag{3.14}$$

$$< l, q_i > = \max_{j=1,\ldots,2M} < l, q_j > \text{ und} \tag{3.15}$$

$$< l, p_2 > = \max_{j=1,2} < l, p_j > . \tag{3.16}$$

Aus 3.16 folgt:

$$\max_{j=1,2} < l, p_j - p_2 > = 0. \tag{3.17}$$

Wegen 3.17 muß nun gelten:

$$< l, p_1 - p_2 > \leq 0.$$

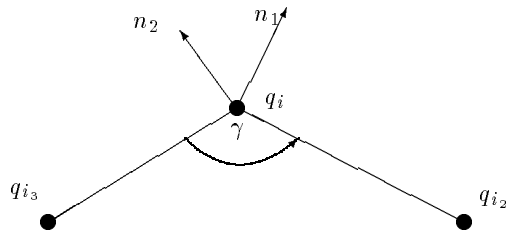
Wir nehmen zunächst an, daß $\beta \in ]\alpha, \alpha + \pi[$. In diesem Fall ist der Winkel $\angle(p_1 - p_2; l) < \frac{\pi}{2}$ und somit $< l, p_1 - p_2 > > 0$. Dies ist ein Widerspruch zu 3.17.
Für $\beta \notin ]\alpha, \alpha + \pi[$ ist der Zwischenwinkel $\angle(p_1 - p_2; l) \geq \frac{\pi}{2}$ und demzufolge $< l, p_1 - p_2 > \leq 0$. Das in 3.14 gewählte $l$ besitzt also folgende Eigenschaft:

$$l = \begin{pmatrix} \cos\beta \\ \sin\beta \end{pmatrix} \text{ mit } \beta \notin ]\alpha, \alpha + \pi[. \tag{3.18}$$

Für eine Richtung $l$ mit dieser Eigenschaft gilt:

$$\max_{j=1,\ldots,2M} < l, q_j - q_i > > 0. \tag{3.19}$$

Um dies zu beweisen, zeigt man, daß für dieses $l$ entweder $< l, q_{i_3} - q_i > > 0$ ist, oder $< l, q_{i_2} - q_i > > 0$ erfüllt ist. Betrachte hierzu nachfolgende Situation, die durch Drehung immer erreichbar ist:




Dabei ist $n_1 = \begin{pmatrix} \cos\delta_1 \\ \sin\delta_1 \end{pmatrix}$ der Normalenvektor auf $\overline{q_{i_2} q_i}$, $n_2 = \begin{pmatrix} \cos\delta_2 \\ \sin\delta_2 \end{pmatrix}$ der Normalenvektor auf $\overline{q_{i_3} q_i}$ und $\gamma$ der orientierte Zwischenwinkel gegen den Uhrzeigersinn zwischen den Vektoren $q_{i_2} - q_i$ und $q_{i_3} - q_i$.
Es folgt:

1. $\alpha < \delta_1$, sonst wäre der Punkt $q_i$ als Punkt $q_{i_2}$ gewählt worden.
2. $\alpha + \pi > \delta_2$, sonst wäre der Punkt $q_i$ als Punkt $q_{i_3}$ gewählt worden.
3. $0 < \gamma < \pi$, denn sonst gäbe es einen Widerspruch zur Wahl der Punkte $q_{i_2}, q_{i_3}, q_i$. Es folgt unter Berücksichtigung von

$$\gamma + (\delta_2 - \delta_1) = \pi \Leftrightarrow \delta_2 - \delta_1 = \pi - \gamma,$$

daß die nachfolgende Beziehung erfüllt ist:

$$\delta_2 > \delta_1.$$

Nehmen wir nun an, daß sowohl $< l, q_{i_3} - q_i > \leq 0$ als auch $< l, q_{i_2} - q_i > \leq 0$ ist, so ergibt sich:

$$\left.\begin{array}{lll} \angle(\vec{q_{i_2}} - \vec{q_i}, l) \geq \frac{\pi}{2} & \Rightarrow & \delta_1 + \pi \geq \beta \geq \delta_1 \\ \angle(\vec{q_{i_3}} - \vec{q_i}, l) \geq \frac{\pi}{2} & \Rightarrow & \delta_2 - \pi \leq \beta \leq \delta_2 \end{array}\right\} \Longrightarrow \beta \in [\delta_1, \delta_2].$$

Dies ist ein Widerspruch zu 3.18, da aufgrund (1)-(3) gilt:

$$[\delta_1, \delta_2] \subset ]\alpha, \alpha + \pi[.$$

Somit folgt die Aussage 3.19. Dieses Resultat widerspricht 3.15. Somit war die Annahme 3.13 falsch, womit die Gültigkeit von 3.6 gezeigt ist. Mit 3.6-3.11 folgt nun:

$$anz(A_{M+1}) = ma - aip = 2(2M) - (4 + (2M-4)) = 4M - 2M = 2M < 2(M+1)$$

Hierbei bezeichnet $ma$ die maximal mögliche Eckenzahl der Menge $A_{M+1}$. Die Anzahl der Punkte, die keine Eckpunkte sind, wird mit $aip$ bezeichnet. $ma$ als Eckenzahl ergäbe sich, falls alle Punkte $q_i + p_j$ $(j = 1, 2;\ i = 1, \ldots, 2M)$ neue Eckpunkte wären. Nach den Resultaten 3.8-3.11 sind jedoch vier Punkte keine Eckpunkte der Menge $A_{M+1}$ und nach 3.6 sowie 3.7 $2M - 4$ Punkte innere Punkte.

Fall 2.2: $\exists\ i_1 < i_2$ mit $i_k \in \{1, \ldots, 2M\}$ $k = 1, 2$ und $\not\exists i_3 \in \{1, \ldots, 2M\} \setminus \{i_1, i_2\}$

mit $q_{i_1} = y(r, A_M)$, $q_{i_2} = y(-r, A_M)$, $q_{i_3} = y(r, A_M)$ oder $q_{i_3} = y(-r, A_M)$

Nun folgt analog zu 3.6 und 3.7:

$$\bigvee_{i_2 > i > i_1} q_i + p_2 \in int(A_M + P)$$
$$\bigvee_{i_2 < i, i < i_1} q_i + p_1 \in int(A_M + P).$$

Somit gilt

$$anz(A_{M+1}) = ma - aip = 2(2M) - (2M-2) = 2(M+1).$$

Fall 2.3: $\exists\, i_1 < i_2 < i_3$ mit $i_k \in \{1,\dots,2M\}, k = 1,2,3$ und
$\nexists i_4 \in \{1,\dots,2M\} \setminus \{i_1, i_2, i_3\}$

$$\text{mit } q_{i_1} = y(r, A_M),\ q_{i_2} = y(r, A_M),\ q_{i_3} = y(-r, A_M),\ \ q_{i_4} = y(-r, A_M).$$

In diesem Fall erhalten wir wie in 3.6, 3.7, 3.8 und 3.9:

$$\begin{aligned}
\bigvee_{i_3 > i > i_2} q_i + p_2 &\in int(A_M + P)\\
\bigvee_{i_3 < i, i < i_1} q_i + p_1 &\in int(A_M + P)\\
q_{i_2} + p_2 &\in co(\{q_{i_2} + p_1, q_{i_1} + p_2\})\\
q_{i_1} + p_1 &\in co(\{q_{i_2} + p_1, q_{i_1} + p_2\}).
\end{aligned}$$

Es ergibt sich auf diese Weise die nachfolgende Abschätzung:

$$anz(A_{M+1}) = ma - aip = 4M - ((2M-3)+2) = 2M + 1 < 2(M+1).$$

Fall 2.4: $\exists\, i_1 < i_2 < i_3$ mit $i_k \in \{1,\dots,2M\},\ k = 1,2,3$ und
$\nexists i_4 \in \{1,\dots,2M\} \setminus \{i_1, i_2, i_3\}$

$$\text{mit } q_{i_1} = y(r, A_M),\ q_{i_2} = y(-r, A_M),\ q_{i_3} = y(-r, A_M),\ \ q_{i_4} = y(r, A_M).$$

In diesem Fall erhalten wir wie in 3.6, 3.7, 3.10 und 3.11:

$$\begin{aligned}
\bigvee_{i_2 > i > i_1} q_i + p_2 &\in int(A_M + P)\\
\bigvee_{i_3 < i, i < i_1} q_i + p_1 &\in int(A_M + P)\\
q_{i_2} + p_2 &\in co(\{q_{i_2} + p_1, q_{i_3} + p_2\})\\
q_{i_3} + p_1 &\in co(\{q_{i_2} + p_1, q_{i_3} + p_2\}).
\end{aligned}$$

Es ergibt sich auf diese Weise die nachfolgende Abschätzung:

$$anz(A_{M+1}) = ma - aip = 4M - ((2M-3)+2) = 2M + 1 < 2(M+1).$$

Insgesamt haben wir somit 3.5 gezeigt und damit den Induktionsschritt. □

**Satz 3.3.2:** *Berechnen wir auf der Grundlage von Summenformeln der Form*

$$\sum_{i=0}^{N} a_i co(F(t_i))$$

*eine Näherung für ein Aumannintegral der Gestalt*

$$\int_a^b F(t)dt = \int_a^b \begin{pmatrix} x_1(t) \\ x_2(t) \end{pmatrix} [c,d]dt \ ,$$

*so benötigen wir dazu einen Zeitaufwand von $O(N^2 \log N)$ für $N \geq 4$.*

*Beweis.* Diese Laufzeitschranke folgt unmittelbar aus Satz 3.3.1 sowie Satz 1.1.4. Sei $t$ die erforderliche Laufzeit und $b_j$ $(j = 1, \ldots, N)$ die Anzahl der Punkte der Menge

$$\sum_{i=0}^{j-1} a_i co(F(t_i)) + a_j co(F(t_j)) =: A + B \ .$$

Die Menge $B$ besitzt höchstens zwei Eckpunkte und die Menge $A$ nach 3.3.1 höchstens $2j$ Ecken. Damit hat die Menge $A + B$ maximal $2 \cdot 2j$ Punkte. Hierbei ist zu berücksichtigen, daß die Menge $A$ bei geeigneter Umsetzung der Summenformel nur noch aus Eckpunkten besteht. Wir benötigen folglich zur Ermittlung der Eckpunkte der konvexen Hülle der Menge $A+B$ einen Aufwand von $O(4j \log 4j)$. Der Zeitaufwand, der zur Lösung des Problems anfällt, setzt sich zusammen aus der Zeit $T_{konv}$, die zur Bestimmung der konvexen Hüllen erforderlich ist, und der benötigten Zeit für das Skalieren $T_{skal}$ und das Addieren $T_{add}$ von Mengen. Somit gilt die Abschätzung:

$$t = T_{konv} + T_{skal} + T_{add} \leq c_1 \sum_{i=1}^{N} b_i \log b_i + c_2(N+1) + c_3 \sum_{i=1}^{N} 2 \cdot 2i$$

$$\leq c_4(N^2 \log 4 + N^2 \log N) + c_2(N+1) + 4c_3N^2 \leq c_5 N^2 \log N.$$

$\diamond$

Wir wenden im folgenden die Treppen-, Trapez- und Simpsonregel zur Approximation von Aumannintegralen auf drei charakteristische Beispiele an:

## 3.4 Beispiel 1

Wir betrachten zunächst das Integral

$$\int\limits_{I=[0,1]} \begin{pmatrix} 0 & e^{\tau-1} \\ e^{2\tau-1} & e^{2\tau-1} \end{pmatrix} [-1,1]^2 d\tau.$$

Dieses Integral ist zu berechnen, falls man die erreichbare Menge zum Zeitpunkt $t = 1$ bei der folgenden linearen Differentialinklusion bestimmen möchte:

$$y'(t) \in \begin{pmatrix} 1 & -1 \\ 4 & -3 \end{pmatrix} y(t) + \begin{pmatrix} 1-t & te^t \\ 3-2t & (2t-1)e^t \end{pmatrix} [-1,1]^2 \; \forall' t \in [0,1],$$

$$y(0) = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

Für das Fundamentalsystem erhalten wir

$$\phi(t,\tau) = \begin{pmatrix} 2(t-\tau)+1 & \tau - t \\ 4(t-\tau) & -(2(t-\tau)-1) \end{pmatrix} e^{\tau - t}.$$

Mit den Bezeichnungen aus Kapitel 0 resultiert:

$$\tilde{F}(t,\tau) * B(\tau) = \begin{pmatrix} (1-t)e^{\tau-t} & te^{2\tau-t} \\ (-2t+3)e^{\tau-t} & (2t-1)e^{2\tau-t} \end{pmatrix}.$$

Betrachten wir den Endzeitpunkt $t = 1$, so muß

$$F(t) = \tilde{F}(1,\tau) = \phi(1,\tau) * B(\tau) = \begin{pmatrix} 0 & e^{2\tau-1} \\ e^{\tau-1} & e^{2\tau-1} \end{pmatrix}$$

sein. Da die exakte Lösung für das Integral nicht bekannt ist, berechnen wir zunächst durch Anwendung der Simpsonregel mit einem Diskretisierungsparameter $N = 20000$ eine Referenzlösung $\tilde{L}$. Für verschiedene Diskretisierungsparameter $N$ approximieren wir die Lösungsmenge und bestimmen jeweils den Hausdorffabstand zu $\tilde{L}$. Wie in 3.2 angegeben, wird die Konvergenzordnung des benutzten Verfahrens abgeschätzt. Da der Aufwand zur Bestimmung des Integrals wesentlich von der Anzahl der Eckpunkte der konvexen Menge abhängig ist, ermitteln wir die Zahl dieser Punkte. Um ein Maß für den anfallenden Zeitaufwand zu bekommen, überlegen wir uns, daß dieser im wesentlichen von der Zeit bestimmt wird, welche bei der Ermittlung der konvexen Hüllen benötigt wird. Die Zeit, die zur Addition oder Skalierung von Mengen gebraucht wird,

sowie die zur Bestimmung der Eckpunkte der Menge $a_i co(F(t_i))$ $(i = 0, \ldots, N)$ lassen wir dabei außer acht. Dies ist möglich, da letztere Menge durch die Angabe ihrer Eckpunkte intern repräsentiert wird. Bezeichnet $N_i$ $(i = 1, \ldots, N-1)$ die Anzahl der Punkte der Menge

$$\sum_{j=0}^{i} a_j co(F(t_j)) + a_{i+1} co(F(t_{i+1})) =: A_i,$$

so ist zur Bestimmung der Eckpunkte der Menge $A_i$ ein Aufwand von $O(N_i \log(N_i))$ erforderlich (vgl. 1.1.4). Da die Ecken der Mengen $A_1, \ldots, A_{N-1}$ zur Ermittlung der approximierenden Lösungsmenge bestimmt werden müssen, ist ein Aufwand von

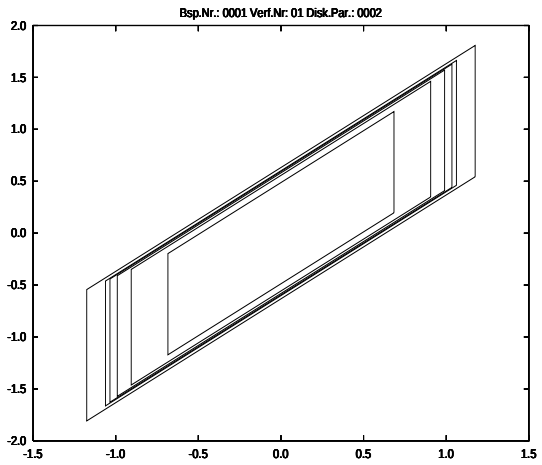
$$\sum_{i=1}^{N-1} O(N_i \log(N_i))$$

erforderlich. Unter Vernachlässigung einer Konstanten berechnen wir nun

$$t = \sum_{i=1}^{N-1} N_i \log(N_i)$$

als ein Maß für die anfallende Zeit.

Tabelle 3.1 gibt die Resultate für den Fall wieder, daß die Treppenregel zur Approximation benutzt wurde.
Für die Diskretisierungsparameter $N = (2, 4, 6, 8, 10)$ ergeben sich die folgenden Lösungsmengen.




Die äußere Menge ist dabei die Referenzlösung und die Innerste, jene die zum Diskretisierungsparameter $N = 2$ gehört. Je größer der Diskretisierungsparameter $N$ gewählt wird, desto näher liegt die zugehörige approximierende Menge

an der Referenzlösung. Größere Parameter werden nicht berücksichtigt, da in diesem Fall die Menge aufgrund der Zeichengenauigkeit mit der Referenzlösung identisch ist.
Wir beobachten, daß alle Näherungslösungen vier Eckpunkte besitzen, und mit Hilfe des berechneten Hausdorffabstandes zu einer Referenzlösung berechnen wir, wie in 3.2 angegeben, für die Treppenregel eine Konvergenzordnung von 1. Diese Konvergenzordnung war zu erwarten, da die Treppenregel bei der Approximation von gewöhnlichen Integralen ebenfalls eine Konvergenzordnung von 1 besitzt, falls z.B. $f \in C^1[a,b]$ ist. Um die Konvergenzordnung theoretisch begründen zu können, müßten wir tiefer in die Theorie der Stützfunktionen einsteigen. In dieser Arbeit wird nicht näher darauf eingegangen. Wir weisen jedoch darauf hin, daß die Konvergenzordnung der einzelnen Verfahren bei diesem Aumannintegral sowie für die nachfolgend betrachteten Integrale in [3] theoretisch begründet wird. Nach [3] muß man, um eine bestimmte Konvergenzordnung zu erreichen, geeignete Forderungen an die Stützfunktion selbst stellen. Diese betreffen vor allem die Differenzierbarkeit der Stützfunktion sowie die Glattheit ihrer Ableitung.

Wie Tabelle 3.2 zu entnehmen ist, besitzt bei Anwendung der Trapezsumme (0.1) jede Approximierende vier Eckpunkte. Darüber hinaus beobachten wir eine Konvergenzordnung von zwei. In diesem Fall ist diese Ordnung wiederum mit Hilfe der Eigenschaften der Stützfunktion sowie deren Ableitung erklärbar.

Wir wissen, daß die Simpsonregel bei der Bestimmung eines Riemannintegrals, falls $f$ auf $[a,b]$ genügend oft differenzierbar ist (vgl. [18]), die Grundlage für ein Verfahren der Ordnung 4 bildet. Im mengenwertigen Fall schätzen wir für obiges Beispiel ebenfalls eine Konvergenzordnung von 4. Die einzelnen Ergebnisse veranschaulicht Tabelle 3.3.

**Bemerkung:** Wesentlich für die Durchführung des Algorithmus von Graham ist die Bestimmung der Determinante aus 2.2.1. Für eine $3 \times 3$ Matrix kann dies zum einen — wie implementiert — mit Hilfe einer QR-Zerlegung nach Householder geschehen ([18], S. 181-184 ) und zum anderen mit der Sarrus-Regel ([12], S. 99). Bei Verwendung der Sarrus-Regel ergibt sich bei Benutzung der Simpson-Regel die Tabelle 3.4. Vergleichen wir diese Ergebnisse mit den Resultaten, die wir durch Anwendung der QR-Zerlegung erhalten haben, so ist zu beobachten, daß sich die berechneten Lösungsmengen hinsichtlich der Genauigkeit in Bezug zu einer Referenzlösung nicht unterscheiden. Die Anzahl der Eckpunkte der approximierenden Mengen ist jedoch im zweiten Fall höher. Hieraus ergibt sich ein höherer Aufwand bei der Bestimmung der Näherungslösung. Der Grund hierfür ist darin zu sehen, daß die Sarrus-Regel bei fast linear abhängigen Zeilen numerisch instabiler ist als die Householder-Zerlegung. Liegen also drei Punkte auf einer Geraden, so erkennen wir dies mit Hilfe der QR-Zerlegung besser. $\diamond$

| Disk.Par. | Ecken | Aufwand | Fehler | Ordnung |
|---|---|---|---|---|
| 0002 | 0004 | 44.36 | 0.803777567151 | ———————– |
| 0004 | 0004 | 133.08 | 0.437848155392 | ———————– |
| 0006 | 0004 | 221.81 | 0.300044198908 | 0.932120599353 |
| 0008 | 0004 | 310.53 | 0.228101244713 | 0.940756314420 |
| 0010 | 0004 | 399.25 | 0.183956120145 | 0.946397132722 |
| 0016 | 0004 | 665.42 | 0.116357072845 | 0.955935862512 |
| 0020 | 0004 | 842.87 | 0.093455105358 | 0.959584086322 |
| 0030 | 0004 | 1286.48 | 0.062631889195 | 0.965107299361 |
| 0040 | 0004 | 1730.10 | 0.047097120075 | 0.968329172986 |
| 0050 | 0004 | 2173.71 | 0.037736837488 | 0.970506249569 |
| 0060 | 0004 | 2617.32 | 0.031480221988 | 0.972106313529 |
| 0070 | 0004 | 3060.94 | 0.027003164573 | 0.973348211199 |
| 0080 | 0004 | 3504.55 | 0.023640971035 | 0.974349721177 |
| 0090 | 0004 | 3948.17 | 0.021023323877 | 0.975180588290 |
| 0100 | 0004 | 4391.78 | 0.018927563253 | 0.975885095028 |
| 0200 | 0004 | 8827.92 | 0.009478568244 | 0.979758782604 |
| 0300 | 0004 | 13264.06 | 0.006322331437 | 0.981539270827 |
| 0400 | 0004 | 17700.21 | 0.004742980807 | 0.982636078580 |
| 0500 | 0004 | 22136.35 | 0.003794976116 | 0.983406280955 |
| 1000 | 0004 | 44317.06 | 0.001898079529 | 0.985432959341 |
| 2000 | 0004 | 88678.48 | 0.000949187632 | 0.987032626981 |
| 3000 | 0004 | 133039.90 | 0.000632824614 | 0.987819006619 |
| 4000 | 0004 | 177401.32 | 0.000474630783 | 0.988322540834 |
| 5000 | 0004 | 221762.74 | 0.000379710541 | 0.988685773117 |
| 10000 | 0004 | 443569.83 | 0.000189861185 | 0.989684140338 |
| 15000 | 0004 | 665376.93 | 0.000126575438 | 0.990191137290 |
| 20000 | 0004 | 887184.03 | 0.000094932071 | 0.990521838007 |
| 100000 | 0004 | 4436097.59 | 0.000018986651 | 0.992026983271 |

Tabelle 3.1: Ergebnisse bei Anwendung der unteren Treppenregel

| Disk.Par. | Ecken | Aufwand | Fehler | Ordnung |
|---|---|---|---|---|
| 0002 | 0004 | 88.72 | 0.145812678941 | ———————— |
| 0004 | 0004 | 177.45 | 0.036880258074 | ———————— |
| 0006 | 0004 | 266.17 | 0.016427107941 | 1.994606901268 |
| 0008 | 0004 | 354.89 | 0.009247343569 | 1.995737855712 |
| 0010 | 0004 | 443.61 | 0.005920405237 | 1.996387646290 |
| 0016 | 0004 | 709.78 | 0.002313550214 | 1.997334217119 |
| 0020 | 0004 | 887.23 | 0.001480803959 | 1.997648505449 |
| 0030 | 0004 | 1330.84 | 0.000658192973 | 1.998078058645 |
| 0040 | 0004 | 1774.46 | 0.000370244944 | 1.998304814704 |
| 0050 | 0004 | 2218.07 | 0.000236960141 | 1.998448939808 |
| 0060 | 0004 | 2661.69 | 0.000164556927 | 1.998550507887 |
| 0070 | 0004 | 3105.30 | 0.000120899531 | 1.998626943115 |
| 0080 | 0004 | 3548.91 | 0.000092563984 | 1.998687134391 |
| 0090 | 0004 | 3992.53 | 0.000073137127 | 1.998736132518 |
| 0100 | 0004 | 4436.14 | 0.000059241161 | 1.998777040000 |
| 0200 | 0004 | 8872.28 | 0.000014810361 | 1.998992514281 |
| 0300 | 0004 | 13308.43 | 0.000006582388 | 1.999086925431 |
| 0400 | 0004 | 17744.57 | 0.000003702595 | 1.999143897167 |
| 0500 | 0004 | 22180.71 | 0.000002369661 | 1.999183432407 |
| 1000 | 0004 | 44361.42 | 0.000000592415 | 1.999285903397 |
| 2000 | 0004 | 88722.84 | 0.000000148104 | 1.999365523965 |
| 3000 | 0004 | 133084.26 | 0.000000065824 | 1.999404362617 |
| 4000 | 0004 | 177445.68 | 0.000000037026 | 1.999429136447 |
| 5000 | 0004 | 221807.10 | 0.000000023697 | 1.999447000818 |
| 10000 | 0004 | 443614.20 | 0.000000005924 | 1.999495623202 |
| 15000 | 0004 | 665421.29 | 0.000000002633 | 1.999520277827 |
| 20000 | 0004 | 887228.39 | 0.000000001481 | 1.999537735412 |
| 100000 | 0004 | 4436141.96 | 0.000000000060 | 1.999177863044 |

Tabelle 3.2: Resultate bei Anwendung der Trapezregel für das erste Beispiel

| Disk.Par. | Ecken | Aufwand | Fehler | Ordnung |
|---|---|---|---|---|
| 0002 | 0004 | 88.72 | 0.008390811720 | ———— |
| 0004 | 0004 | 177.45 | 0.000570052178 | ———— |
| 0006 | 0004 | 266.17 | 0.000114425969 | 3.960389709539 |
| 0008 | 0004 | 354.89 | 0.000036410923 | 3.968650758838 |
| 0010 | 0004 | 443.61 | 0.000014953226 | 3.973412248731 |
| 0016 | 0004 | 709.78 | 0.000002288212 | 3.980364864303 |
| 0020 | 0004 | 887.23 | 0.000000937871 | 3.982676854783 |
| 0030 | 0004 | 1330.84 | 0.000000185379 | 3.985838904041 |
| 0040 | 0004 | 1774.46 | 0.000000058669 | 3.987508833733 |
| 0050 | 0004 | 2218.07 | 0.000000024033 | 3.988570320493 |
| 0060 | 0004 | 2661.69 | 0.000000011591 | 3.989318272640 |
| 0070 | 0004 | 3105.30 | 0.000000006257 | 3.989880930067 |
| 0080 | 0004 | 3548.91 | 0.000000003668 | 3.990323643059 |
| 0090 | 0004 | 3992.53 | 0.000000002290 | 3.990683644035 |
| 0100 | 0004 | 4436.14 | 0.000000001502 | 3.990983850414 |
| 0200 | 0004 | 8872.28 | 0.000000000094 | 3.992521980482 |
| 0300 | 0004 | 13308.43 | 0.000000000019 | 3.993023783730 |
| 0400 | 0004 | 17744.57 | 0.000000000006 | 3.992917682324 |
| 0500 | 0004 | 22180.71 | 0.000000000002 | 3.992260423652 |
| 0600 | 0004 | 26616.85 | 0.000000000001 | 3.990665739627 |
| 0700 | 0004 | 31052.99 | 0.000000000001 | 3.988420427895 |
| 0800 | 0004 | 35489.14 | 0.000000000000 | 3.984704742174 |
| 0900 | 0004 | 39925.28 | 0.000000000000 | 3.978405553701 |
| 1000 | 0004 | 44361.42 | 0.000000000000 | 3.971789589987 |

Tabelle 3.3: Resultate bei Anwendung der Simpsonregel für das erste Beispiel

| Disk.Par. | Ecken | Aufwand | Fehler | Ordnung |
|---|---|---|---|---|
| 0002 | 0005 | 88.72 | 0.008390811720 | ———— |
| 0004 | 0007 | 243.99 | 0.000570052178 | ———— |
| 0006 | 0005 | 379.37 | 0.000114425969 | 3.960389709539 |
| 0008 | 0006 | 386.00 | 0.000036410923 | 3.968650758838 |
| 0010 | 0004 | 617.92 | 0.000014953226 | 3.973412248731 |
| 0016 | 0008 | 1251.59 | 0.000002288212 | 3.980364864303 |
| 0020 | 0008 | 1544.83 | 0.000000937871 | 3.982676854783 |
| 0030 | 0010 | 2512.78 | 0.000000185379 | 3.985838904041 |
| 0040 | 0007 | 3825.76 | 0.000000058669 | 3.987508833733 |
| 0050 | 0008 | 4867.44 | 0.000000024033 | 3.988570320493 |
| 0060 | 0010 | 6009.09 | 0.000000011591 | 3.989318272640 |
| 0070 | 0007 | 7186.28 | 0.000000006257 | 3.989880930067 |
| 0080 | 0009 | 9173.62 | 0.000000003668 | 3.990323643059 |
| 0090 | 0009 | 9901.95 | 0.000000002290 | 3.990683706328 |
| 0100 | 0012 | 11306.16 | 0.000000001502 | 3.990983850414 |
| 0200 | 0010 | 30626.78 | 0.000000000094 | 3.992521980482 |
| 0300 | 0009 | 46865.98 | 0.000000000019 | 3.993023783730 |
| 0400 | 0011 | 65023.10 | 0.000000000006 | 3.992934056584 |
| 0500 | 0013 | 88333.43 | 0.000000000002 | 3.992298373564 |
| 0600 | 0011 | 113871.85 | 0.000000000001 | 3.990515569950 |
| 0700 | 0012 | 129808.98 | 0.000000000001 | 3.988154102041 |
| 0800 | 0012 | 153837.33 | 0.000000000000 | 3.984704742174 |
| 0900 | 0013 | 182952.14 | 0.000000000000 | 3.978405553701 |
| 1000 | 0014 | 210628.15 | 0.000000000000 | 3.970848748975 |

Tabelle 3.4: Ergebnisse bei Verwendung der Simpsonregel zur Approximation des Integrals und der Sarrus-Regel für die Ermittlung der Determinaten

## 3.5 Beispiel 2

Im folgenden wenden wir die drei Quadraturformel auf ein Beispiel an, das von Veliov in [20] vorgestellt worden ist. Dabei wird sich herausstellen, daß die Simpsonregel nur noch eine Konvergenzordnung von 2 besitzt. Wir betrachten das lineare Kontrollproblem:

$$y'(t) \in \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} y(t) + \begin{pmatrix} 0 \\ 1 \end{pmatrix} [-1,1] \qquad \forall' t \in [0,1],$$

$$y(0) = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

Als Fundamentalsystem ergibt sich

$$\phi(t,\tau) = \begin{pmatrix} 1 & t-\tau \\ 0 & 1 \end{pmatrix},$$

und die erreichbare Menge zum Zeitpunkt $t = 1$ errechnet sich aus

$$\int_0^1 \begin{pmatrix} 1-\tau \\ 1 \end{pmatrix} [-1,1] d\tau.$$

Mit Hilfe geeigneter Konvergenzsätze (vgl. [3]) können wir aufgrund der Eigenschaften der Stützfunktion sowie deren Ableitungen bei diesem Beispiel höchstens eine Konvergenz zweiter Ordnung erwarten. In den Tabellen 3.5, 3.6 und 3.7 finden wir die Resultate bei Verwendung der Treppen-, Trapez- und Simpsonregel. Die Referenzlösung ist mit Hilfe der Trapezregel und einem Diskretisierungsparameter von $N = 2000$ bestimmt worden. Hierbei ist zu beobachten, daß die obere Schranke für die Anzahl der Eckpunkte, wie sie in 3.3.1 theoretisch vorhergesagt worden ist, bei Verwendung der angegebenen Regeln angenommen wird. Als Lösungsmengen berechnen wir mit Hilfe der Simpsonregel und den Diskretisierungsparametern $N = 2, 4, 6, 8, 10$ und 100 die in Abbildung 3.1 dargestellten konvexen Mengen. Die Treppenregel liefert für dieses Beispiel ein Verfahren der Ordnung 1 und die Trapezsumme eines der Ordnung 2. Bei Verwendung der Simpsonregel ergibt sich eine Reduktion der Konvergenzordnung von 4 auf 2. Dies ist damit zu begründen, daß die Stützfunktion

$$\delta^*(l, \phi(1,\tau) \begin{pmatrix} 0 \\ 1 \end{pmatrix} [-1,1]) = |(1-\tau, 1)l|$$

nur absolut stetig ist, und ihre Ableitung von beschränkter Variation gleichmäßig für alle $l \in \mathbb{R}^n$ mit $\|l\|_2 = 1$ ist.

| Disk.Par. | Ecken | Aufwand | Fehler | Ordnung |
|---|---|---|---|---|
| 0002 | 0004 | 5.55 | 0.250000000000 | ———————– |
| 0004 | 0008 | 52.00 | 0.125000000000 | ———————– |
| 0006 | 0012 | 156.28 | 0.083333333333 | 1.000000000000 |
| 0008 | 0016 | 325.85 | 0.062500000000 | 1.000000000000 |
| 0010 | 0020 | 565.76 | 0.050000000000 | 1.000000000000 |
| 0016 | 0032 | 1742.18 | 0.031250000000 | 1.000000000000 |
| 0020 | 0040 | 2932.33 | 0.025000000000 | 1.000000000000 |
| 0030 | 0060 | 7432.36 | 0.016666666667 | 1.000000000000 |
| 0040 | 0080 | 14236.77 | 0.012500000000 | 0.999999999999 |
| 0050 | 0100 | 23464.05 | 0.010000000000 | 0.999999999999 |
| 0060 | 0120 | 35205.28 | 0.008333333333 | 0.999999999999 |
| 0070 | 0140 | 49534.48 | 0.007142857143 | 0.999999999999 |
| 0080 | 0160 | 66514.03 | 0.006250000000 | 0.999999999999 |
| 0090 | 0180 | 86197.88 | 0.005555555556 | 0.999999999999 |
| 0100 | 0200 | 108633.53 | 0.005000000000 | 0.999999999999 |
| 0200 | 0400 | 492097.85 | 0.002500000000 | 0.999999999998 |
| 0300 | 0600 | 1181962.68 | 0.001666666667 | 0.999999999997 |
| 0400 | 0800 | 2194983.64 | 0.001250000000 | 0.999999999997 |
| 0500 | 1000 | 3542853.39 | 0.001000000000 | 0.999999999996 |
| 0600 | 1200 | 5234584.55 | 0.000833333333 | 0.999999999996 |
| 0700 | 1400 | 7277518.06 | 0.000714285714 | 0.999999999996 |
| 0800 | 1600 | 9677849.57 | 0.000625000000 | 0.999999999993 |
| 0900 | 1800 | 12440940.00 | 0.000555555556 | 0.999999999993 |
| 1000 | 2000 | 15571514.48 | 0.000500000000 | 0.999999999994 |
| 2000 | 4000 | 67861629.31 | 0.000250000000 | 0.999999999998 |

Tabelle 3.5: Approximation des zweiten Aumannintegrals mittels der unteren Treppenregel

| Disk.Par. | Ecken | Aufwand | Fehler | Ordnung |
|---|---|---|---|---|
| 0002 | 0006 | 22.18 | 0.060684671912 | ———— |
| 0004 | 0010 | 96.36 | 0.015504380487 | ———— |
| 0006 | 0014 | 232.55 | 0.006919088667 | 1.989908661716 |
| 0008 | 0018 | 436.75 | 0.003898334983 | 1.991745909320 |
| 0010 | 0022 | 713.32 | 0.002496688656 | 1.992988923846 |
| 0016 | 0034 | 2008.35 | 0.000976018156 | 1.994812042602 |
| 0020 | 0042 | 3282.90 | 0.000624733623 | 1.995449774503 |
| 0030 | 0062 | 8006.86 | 0.000277273197 | 1.997056206829 |
| 0040 | 0082 | 15048.79 | 0.000156204150 | 1.996761850426 |
| 0050 | 0102 | 24523.72 | 0.000099719961 | 1.998042039817 |
| 0060 | 0122 | 36520.64 | 0.000069353518 | 1.997622127529 |
| 0070 | 0142 | 51112.22 | 0.000050802232 | 1.998789680435 |
| 0080 | 0162 | 68359.90 | 0.000038999230 | 1.997954229966 |
| 0090 | 0182 | 88316.88 | 0.000030713205 | 1.999086103690 |
| 0100 | 0202 | 111030.11 | 0.000024923109 | 1.998549420309 |
| 0200 | 0402 | 497445.54 | 0.000006215993 | 1.999413693727 |
| 0300 | 0602 | 1190470.77 | 0.000002754329 | 2.000168578653 |
| 0400 | 0802 | 2206788.05 | 0.000001499970 | 2.007185862918 |
| 0500 | 1002 | 3558055.20 | 0.000000982943 | 2.001958195292 |
| 0600 | 1202 | 5253264.29 | 0.000000679489 | 2.002798449426 |
| 0700 | 1402 | 7299742.71 | 0.000000494908 | 2.004393150389 |
| 0800 | 1602 | 9703676.47 | 0.000000373583 | 2.006956466541 |
| 0900 | 1802 | 12470419.28 | 0.000000294070 | 2.007498598483 |
| 1000 | 2002 | 15604690.68 | 0.000000242657 | 2.003995695630 |

Tabelle 3.6: Approximation des zweiten Aumannintegrals mit der Trapezsumme

| Disk.Par. | Ecken | Aufwand | Fehler | Ordnung |
|---|---|---|---|---|
| 0002 | 0006 | 22.18 | 0.073514219909 | ———— |
| 0004 | 0010 | 96.36 | 0.020188477327 | ———— |
| 0006 | 0014 | 232.55 | 0.008827717043 | 2.040164640876 |
| 0008 | 0018 | 436.75 | 0.004949257154 | 2.028248193673 |
| 0010 | 0022 | 713.32 | 0.003201757414 | 2.009637440798 |
| 0016 | 0034 | 2008.35 | 0.001279272993 | 1.990068016993 |
| 0020 | 0042 | 3282.90 | 0.000811936509 | 1.996625728871 |
| 0030 | 0062 | 8006.86 | 0.000355947930 | 2.004107787266 |
| 0040 | 0082 | 15048.79 | 0.000203736844 | 1.996033989146 |
| 0050 | 0102 | 24523.72 | 0.000126859769 | 2.007256403155 |
| 0060 | 0122 | 36520.64 | 0.000089862356 | 1.999441580779 |
| 0070 | 0142 | 51112.22 | 0.000061640565 | 2.023459342399 |
| 0080 | 0162 | 68359.90 | 0.000051092239 | 1.995917570507 |
| 0090 | 0182 | 88316.88 | 0.000038523886 | 2.011099422155 |
| 0100 | 0202 | 111030.11 | 0.000031470255 | 2.008099978489 |
| 0200 | 0402 | 497445.54 | 0.000008137342 | 1.998046458854 |
| 0300 | 0602 | 1190470.77 | 0.000003487865 | 2.006624491653 |
| 0400 | 0802 | 2206788.05 | 0.000002030304 | 1.998771238850 |
| 0500 | 1002 | 3558055.20 | 0.000001321518 | 1.995331433035 |
| 0600 | 1202 | 5253264.29 | 0.000000876228 | 2.004735045383 |
| 0700 | 1402 | 7299742.71 | 0.000000731465 | 1.979863790446 |
| 0800 | 1602 | 9703676.47 | 0.000000556133 | 1.981688468196 |
| 0900 | 1802 | 12470419.28 | 0.000000410002 | 1.994878176511 |
| 1000 | 2002 | 15604690.68 | 0.000000320047 | 2.001672448684 |

Tabelle 3.7: Approximation des zweiten Aumannintegrals mit der Simpsonregel

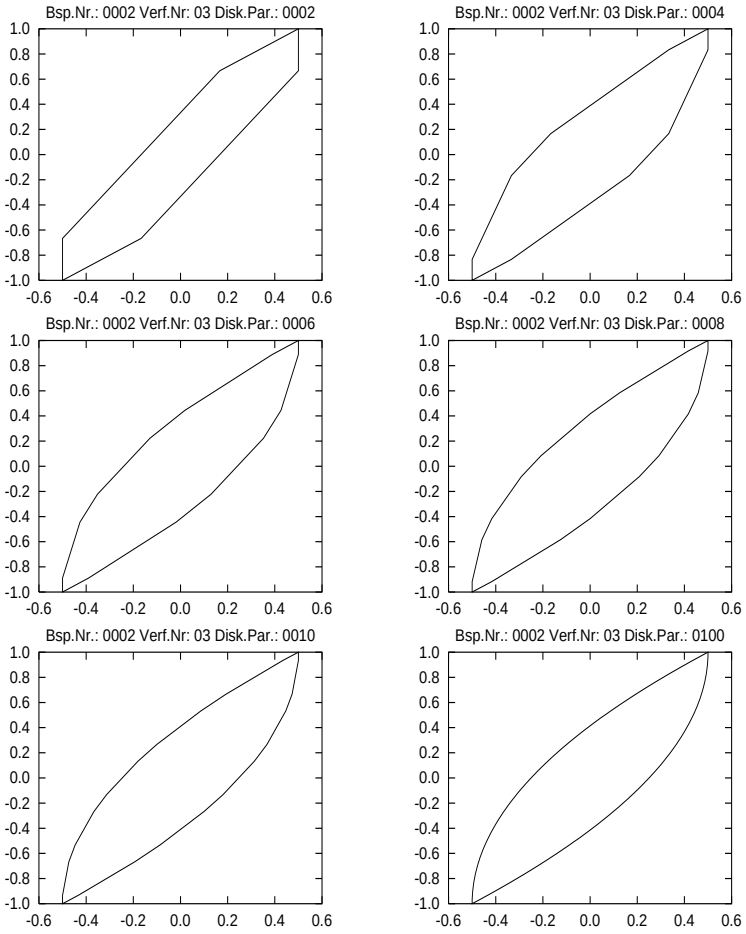



Abbildung 3.1: Lösungsmengen für die Diskretisierungsparameter $N = 2, 4, 6, 8, 10, 100$

## 3.6 Beispiel 3

Wir betrachten nun ein Beispiel, bei dem alle Verfahren die gleiche Konvergenzordnung besitzen, und bei der die Anzahl der Ecken der approximierenden Lösungsmenge jeweils dem Diskretisierungsparameter entspricht.

Dieses Beispiel wurde von Veliov in [21] diskutiert. Er begründet in dieser Veröffentlichung geometrisch, warum bei Verwendung der Simpsonregel lediglich eine Konvergenzordnung von 2 zu erwarten ist. Wir untersuchen im folgenden das Integral

$$\int_0^{2\pi} A(t)[-1,1]dt = \int_0^{2\pi} \begin{pmatrix} sin(t) \\ cos(t) \end{pmatrix} [-1,1]dt.$$

Als exakte Lösung für das Aumannintegral erhalten wir den Kreis $4B_1(0)$. Bei der Berechnung des Fehlers muß folglich keine Referenzmenge bestimmt werden. Damit werden unsere geschätzten Fehlerwerte genauer. Der Fehler läßt sich nun wie folgt berechnen:

$$haus(P,Q) = \sup_{\|l\|_2=1} |\delta^*(l,P) - \delta^*(l,Q)| = \sup_{\|l\|_2=1} |\delta^*(l,P) - \delta^*(l,4B_1(0))| =$$

$$\sup_{\|l\|_2=1} |\delta^*(l,P) - 4\|l\|_2| \approx \max_{j=0,\ldots,M-1} |\max_{i=1,\ldots,r} < l^j, p_i > -4.0|.$$

Dabei sind die $l^j$ $(j = 0,\ldots,M-1)$ wie in 3.1.3 beschrieben gewählt. Bei diesem Beispiel wollen wir uns zunächst einmal anschauen, welche Mengen im Zuge einer Approximation auftreten. Wir bestimmen dazu eine Näherung mit Hilfe der Trapez- bzw. Simpsonregel und einem Diskretisierungsparameter von $N = 10$. In diesen Fällen sind jeweils zehn Mengenadditionen erforderlich. Die Abbildungen 3.2 und 3.3 veranschaulichen, welche Mengen bei der i-ten $(i = 1,\ldots,10)$ durchgeführten Mengensummation addiert werden, und welche Menge sich in diesem Schritt aus der Addition ergibt. Wir beobachten, daß in jedem Schritt zu einer konvexen Menge eine Strecke addiert wird, welche durch den Ursprung geht. Auf diese Weise resultiert eine Menge, die die vorherige Menge enthält. Die Menge bläht sich Schritt für Schritt immer weiter auf, bis die Menge im letzten Schritt skaliert wird. Benutzen wir die Simpsonregel zur Approximation des Aumannintegrales und variieren die Diskretisierungsparameter, so ergeben sich für die Parameter $N = 2,4,6,8,10,100$ die in Abbildung 3.4 dargestellten Mengen. Die Tabellen 3.8, 3.9 und 3.10 geben Auskunft über die numerischen Ergebnisse.

**Bemerkung:** Bei diesem Beispiel beobachten wir sowohl für die Treppenregel als auch für die Trapezregel eine Konvergenzordnung von 2. Die Konvergenzordnung von 2 für die Treppenregel ist damit zu begründen, daß bei diesem

| Disk.Par. | Ecken | Aufwand | Fehler | Ordnung |
|---|---|---|---|---|
| 0002 | 0002 | 5.55 | 4.000000000000 | ———————– |
| 0004 | 0004 | 38.82 | 0.858407346410 | ———————– |
| 0006 | 0006 | 111.64 | 0.372401271532 | 2.059626766937 |
| 0008 | 0008 | 229.45 | 0.207762204126 | 2.046729198401 |
| 0010 | 0010 | 395.93 | 0.132468803814 | 2.039452664683 |
| 0016 | 0016 | 1213.08 | 0.051536796109 | 2.028993833006 |
| 0020 | 0020 | 2041.31 | 0.032952924981 | 2.025551092747 |
| 0030 | 0030 | 5181.43 | 0.014632336938 | 2.020863917368 |
| 0040 | 0040 | 9943.58 | 0.008228054583 | 2.018396261729 |
| 0050 | 0050 | 16416.20 | 0.005265174909 | 2.016829636450 |
| 0060 | 0060 | 24667.33 | 0.003656077313 | 2.015726276844 |
| 0070 | 0070 | 34752.34 | 0.002685967333 | 2.014896247099 |
| 0080 | 0080 | 46717.88 | 0.002056379006 | 2.014242771568 |
| 0090 | 0090 | 60604.32 | 0.001624758225 | 2.013710904545 |
| 0100 | 0100 | 76447.23 | 0.001316033847 | 2.013266913540 |
| 0200 | 0200 | 348292.11 | 0.000328992225 | 2.010928846532 |
| 0300 | 0300 | 839169.09 | 0.000146217430 | 2.009904610414 |
| 0400 | 0400 | 1561635.73 | 0.000082247042 | 2.009286570200 |
| 0500 | 0500 | 2524457.19 | 0.000052638029 | 2.008857692119 |
| 1000 | 1000 | 11143755.85 | 0.000013159481 | 2.007746082141 |
| 2000 | 2000 | 48747713.04 | 0.000003289869 | 2.006882199991 |

Tabelle 3.8: Ergebnisse bei Anwendung der unteren Treppenregel beim dritten Beispiel

speziellen Integral Treppenregel und Trapezregel identisch sind. Beachten wir, daß $F(0) = F(2\pi)$, so gilt:

$$A_{Trapezregel} = \frac{h}{2}(F(0) + F(2\pi) + 2\sum_{i=1}^{N-1} F(t_i)) =$$

$$hF(0) + h\sum_{i=1}^{N-1} F(t_i) = h \cdot \sum_{i=0}^{N-1} F(t_i) = A_{Treppenregel}.$$

$\diamond$

| Disk.Par. | Ecken | Aufwand | Fehler | Ordnung |
|---|---|---|---|---|
| 0002 | 0002 | 11.09 | 4.000000000000 | ———————– |
| 0004 | 0004 | 55.45 | 0.858407346410 | ———————– |
| 0006 | 0006 | 141.46 | 0.372401271532 | 2.059626766937 |
| 0008 | 0008 | 273.81 | 0.207762204126 | 2.046729198401 |
| 0010 | 0010 | 455.85 | 0.132468803814 | 2.039452664683 |
| 0016 | 0016 | 1323.98 | 0.051536796109 | 2.028993833006 |
| 0020 | 0020 | 2188.87 | 0.032952924981 | 2.025551092747 |
| 0030 | 0030 | 5427.09 | 0.014632336938 | 2.020863917368 |
| 0040 | 0040 | 10294.14 | 0.008228054583 | 2.018396261729 |
| 0050 | 0050 | 16876.71 | 0.005265174909 | 2.016829636450 |
| 0060 | 0060 | 25241.83 | 0.003656077313 | 2.015726276844 |
| 0070 | 0070 | 35444.17 | 0.002685967333 | 2.014896247099 |
| 0080 | 0080 | 47529.91 | 0.002056379006 | 2.014242771568 |
| 0090 | 0090 | 61539.06 | 0.001624758225 | 2.013710904545 |
| 0100 | 0100 | 77506.89 | 0.001316033847 | 2.013266913539 |
| 0200 | 0200 | 350688.70 | 0.000328992225 | 2.010928846531 |
| 0300 | 0300 | 843007.25 | 0.000146217430 | 2.009904610417 |
| 0400 | 0400 | 1566983.42 | 0.000082247042 | 2.009286570200 |
| 0500 | 0500 | 2531364.94 | 0.000052638029 | 2.008857692133 |
| 1000 | 1000 | 11158957.66 | 0.000013159481 | 2.007746082178 |
| 2000 | 2000 | 48780889.24 | 0.000003289869 | 2.006882199947 |

Tabelle 3.9: Numerische Resultate bei Anwendung der Trapezregel beim dritten Beispiel

| Disk.Par. | Ecken | Aufwand | Fehler | Ordnung |
|---|---|---|---|---|
| 0002 | 0002 | 11.09 | 4.000000000000 | ———————– |
| 0004 | 0004 | 55.45 | 1.905604897607 | ———————– |
| 0006 | 0006 | 141.46 | 0.372401271532 | 4.026444634180 |
| 0008 | 0008 | 273.81 | 0.424643918221 | 2.165923533351 |
| 0010 | 0010 | 455.85 | 0.132468803814 | 2.909783435725 |
| 0016 | 0016 | 1323.98 | 0.103611932115 | 2.100493482722 |
| 0020 | 0020 | 2188.87 | 0.066124884592 | 2.088312694677 |
| 0030 | 0030 | 5427.09 | 0.014632336938 | 2.416652702754 |
| 0040 | 0040 | 10294.14 | 0.016469678049 | 2.063347750711 |
| 0050 | 0050 | 16876.71 | 0.005265174909 | 2.332570609245 |
| 0060 | 0060 | 25241.83 | 0.007314830521 | 2.054116611612 |
| 0070 | 0070 | 35444.17 | 0.002685967333 | 2.293519604559 |
| 0080 | 0080 | 47529.91 | 0.004113604198 | 2.048999920596 |
| 0090 | 0090 | 61539.06 | 0.001624758225 | 2.269844547702 |
| 0100 | 0100 | 77506.89 | 0.002632414201 | 2.045637593557 |
| 0200 | 0200 | 350688.70 | 0.000658006099 | 2.037589205379 |
| 0300 | 0300 | 843007.25 | 0.000292439137 | 2.034065468236 |
| 0400 | 0400 | 1566983.42 | 0.000164495436 | 2.031939503662 |
| 0500 | 0500 | 2531364.94 | 0.000105276612 | 2.030464319312 |
| 1000 | 1000 | 11158957.66 | 0.000026318997 | 2.026640995000 |
| 2000 | 2000 | 48780889.24 | 0.000006579740 | 2.023669825595 |

Tabelle 3.10: Numerische Resultate bei Anwendung der Simpsonregel zur Approximation des Aumannintegrals beim dritten Beispiel

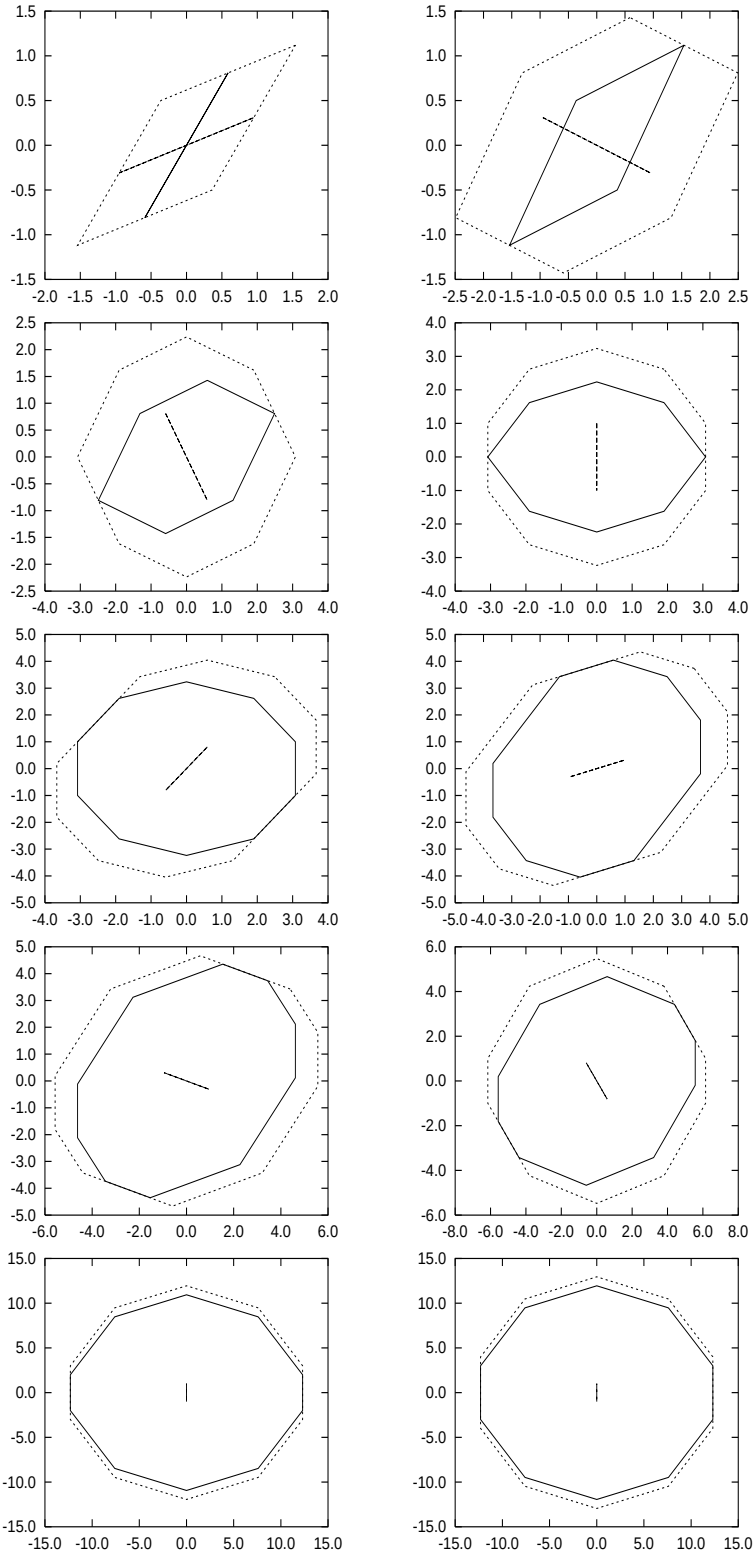


Abbildung 3.2: Mengensummationen, die bei der Approximation mit Hilfe der Trapezregel für eine Diskretisierungsparameter von $N = 10$ auftreten

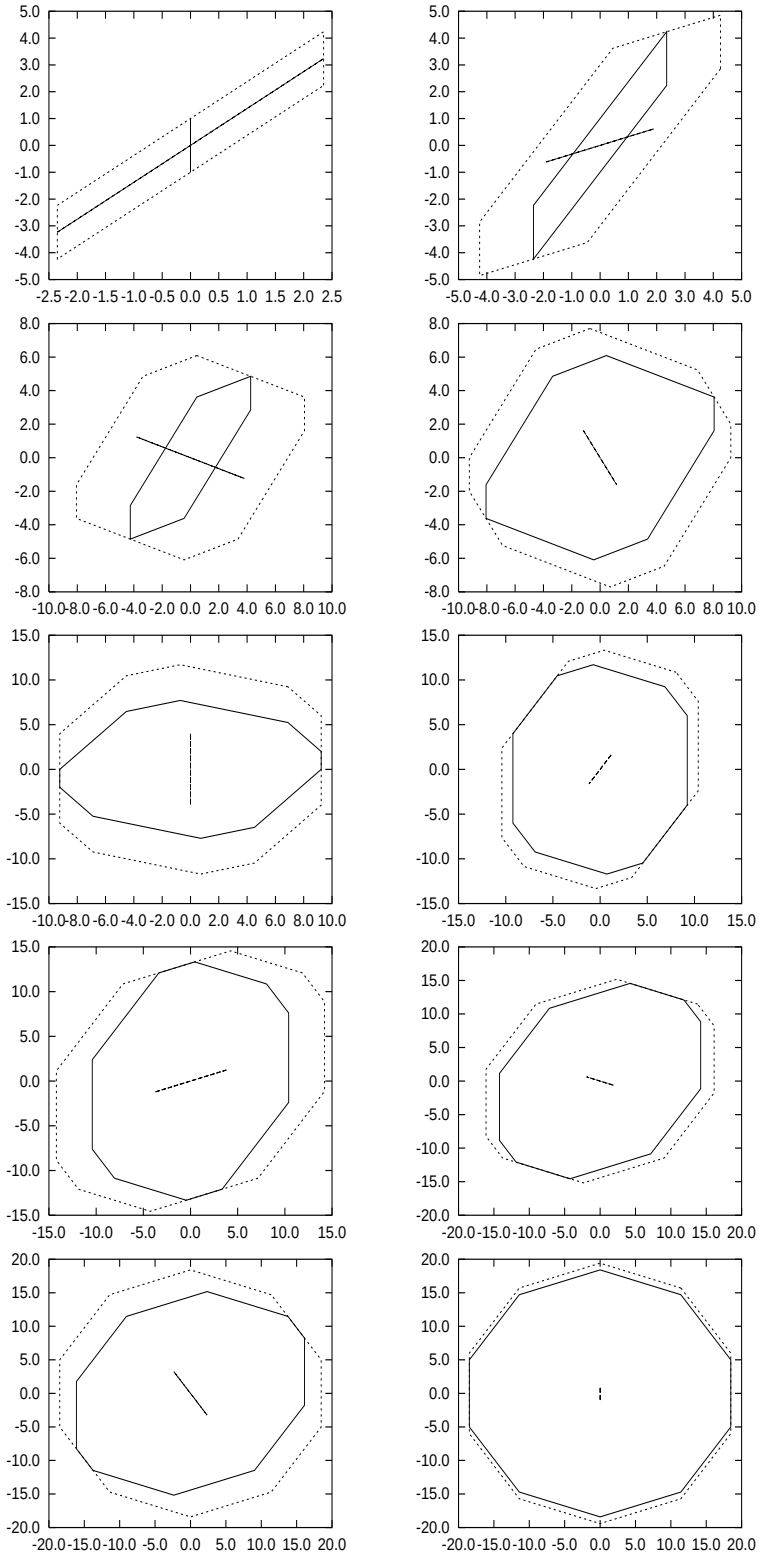


Abbildung 3.3: Mengensummationen, die bei der Approximation mit Hilfe der Simpsonregel für eine Diskretisierungsparameter von $N = 10$ auftreten

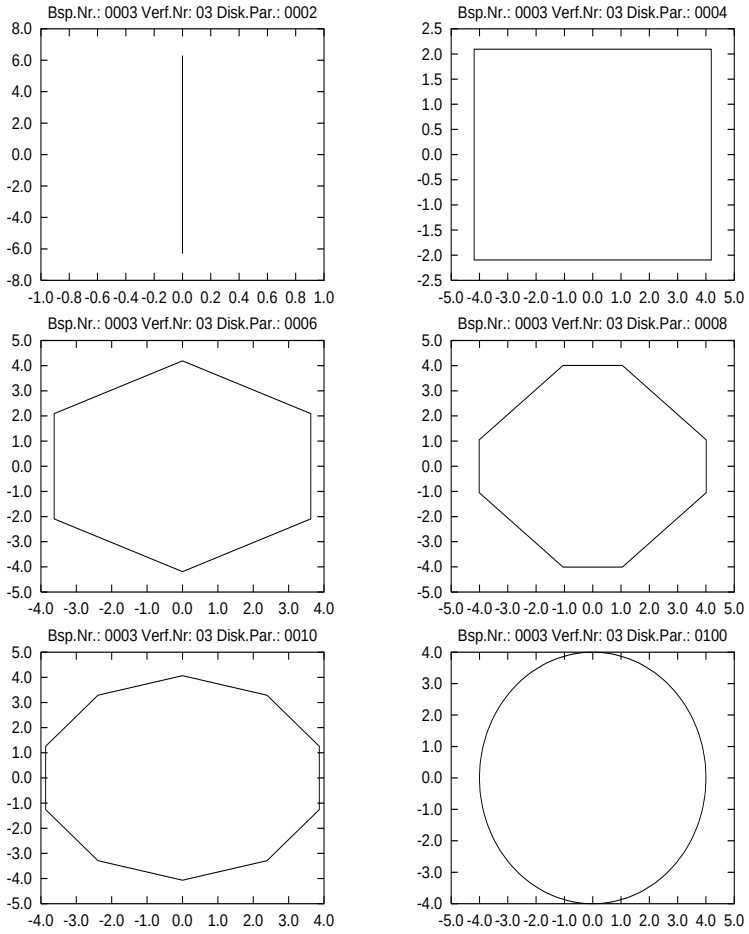



Abbildung 3.4: Approximierende Mengen bei Verwendung der Simpsonregel und den Diskretisierungsparametern $N = 2, 4, 6, 8, 10, 100$

# Kapitel 4

# Ermittlung konvexer Hüllen im $\mathbb{R}^n$

In Kapitel 1 haben wir uns ausführlich mit Algorithmen beschäftigt, welche es ermöglichen, die konvexe Hülle einer endlichen Punktemenge aus der Ebene zu bestimmen. In diesem Kapitel werden nun Algorithmen behandeln, die die konvexe Hülle von endlich vielen Punkten aus dem $\mathbb{R}^n$ berechnen können.

Nach [7] benötigt ein optimaler Algorithmus zur Bestimmung der konvexen Hülle von $N$ Punkten aus dem $\mathbb{R}^n$ $O(N \log N + N^{\lfloor n/2 \rfloor})$ Zeit. Dabei ist $\lfloor n/2 \rfloor$ die größte ganze Zahl kleiner oder gleich $n/2$.

Für die Fälle $n = 2, 3$ wurden schon vor längerer Zeit Algorithmen zur Lösung des Problems entwickelt (vgl. Kapitel 2). In höheren Dimensionen blieb das Problem längere Zeit ungelöst. Vor gut zehn Jahren wurde von Seidel in [16] ein Algorithmus vorgestellt, der dieses Problem für gerades $n$ mit einem optimalen Zeitaufwand lösen konnte. Später entwickelte er einen Algorithmus für beliebige Dimensionen $n$ [17]. Dieser benötigte einen Zeitaufwand von $O(N^{\lfloor n/2 \rfloor} \log N)$. Erst 1993 wurde von Bernhard Chazelle in [7] ein Algorithmus veröffentlicht, der für beliebige Dimensionen $n$ optimales Laufzeitverhalten besitzt.

Im folgenden stellen wir zwei Ansätze vor. Diese bilden in leichten Variationen die Grundlage für die meisten bisher entwickelten Algorithmen zur Ermittlung der konvexen Hülle einer endlichen Anzahl von Punkten aus dem $\mathbb{R}^n$.

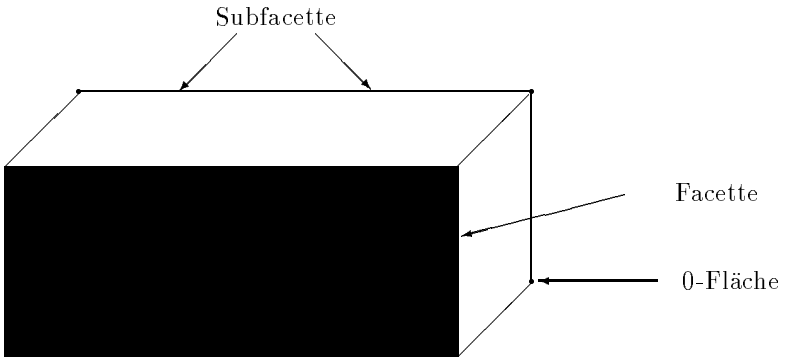
## 4.1 Grundlegende Definitionen

Zur Beschreibung der Ansätze müssen wir zunächst einige Begriffe erläutern.

**Definition 4.1.1:**
Sei $M_N := \{p_i \in \mathbb{R}^n | i = 1, \ldots, N\}$ eine Menge von $N$ $(N \in \mathbb{N})$ Punkten aus dem $\mathbb{R}^n$. Im folgenden bezeichnen wir $co(M_N)$ häufig als Polytop $P$.
Ein Punkt $p$ aus $co(M_N)$ ist eine *Ecke*, falls es keine zwei verschiedene Punkte

$x \in M_N$ und $y \in M_N$ gibt, sodaß $p \in co(\{x, y\})$.
Eine *k-Fläche* eines Polytops $P$ ist die konvexe Hülle von (k+1) verschiedenen Ecken aus $P$ $(k \leq n-1)$. Zusätzlich bezeichnen wir eine (n-1)-Fläche als *Facette*, eine (n-2)-Fläche als *Subfacette*, eine 1-Fläche als *Kante* und die 0-Fläche als *Punkt*. Die leere Menge ist eine (-1)-Fläche. ◇

**Beispiel 4.1.2:** Im folgenden Quader sind die Begriffe 0-Fläche, Subfacette und Facette veranschaulicht. Hierbei ist zu beachten, daß im $\mathbb{R}^3$ 1-Flächen und Subfacetten identisch sind.



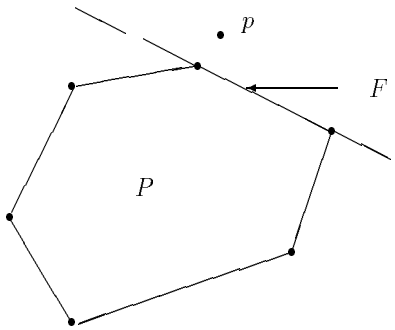

**Definition 4.1.3:**
Sei $M_N := \{p_i \in \mathbb{R}^n | i = 1, \ldots, N\}$ eine endliche Punktemenge.
Die Punkte $p_1, \ldots, p_N$ sind *affin unabhängig*, falls die (N-1) Vektoren $p_2 - p_1, \ldots, p_N - p_1$ linear unabhängig sind. Sind $p_1, \ldots, p_N$ affin unabhängig, so bilden sie die *affine Basis* von $M_N$. Ein *d-Simplex* ist die konvexe Hülle von (d+1) affin unabhängigen Punkten. ◇

**Beispiel 4.1.4:** In nachfolgender Tabelle findet man in Abhängigkeit von der Raumdimension $n$ die Figur, die ein n-Simplex darstellt.

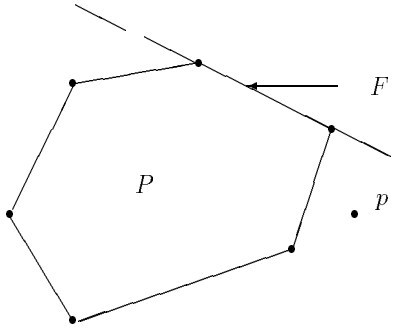
| **Dimension** | **Gebilde** |
|---|---|
| 0 | Punkt |
| 1 | Gerade |
| 2 | Dreieck |
| 3 | Tetraeder |

**Definition 4.1.5:**
Sei $P \in \mathbb{R}^n$ ein Polytop.
Ein Punkt $p \in \mathbb{R}^n$ liegt *diesseits* einer Facette $F \subset P$, falls die affine Hülle $aff(F)$ den Raum so in zwei Halbräume teilt, daß $p$ und alle Flächen von $P$ im gleichen Halbraum liegen. Ist dies nicht der Fall, so liegt $p$ *jenseits* von $F$.
◇

**Beispiel 4.1.6:** Für $n = 2$ veranschaulicht nachfolgende Grafik die Begriffe diesseits und jenseits liegend. Die Facette $F$ ist in diesem Fall die Verbindungsgerade zwischen zwei Punkten aus der Ebene.




$p$ liegt jenseits der Facette $F$




$p$ liegt diesseits der Facette $F$

Die Algorithmen aus Kapitel 1 haben eine vollständige Beschreibung der Umrandung geliefert, indem die Eckpunkte sortiert ausgegeben worden sind. Auf diese Weise konnten die konvexen Mengen leicht gezeichnet werden. In höheren Dimensionen genügt es nun nicht mehr, die Punkte nach einem Sortierkriterium zu sortieren und dann auszugeben. Die meisten Algorithmen für höhere Dimensionen erzeugen eine komplette Beschreibung aller Flächen. Damit nimmt die Komplexität und der Aufwand zur Speicherung der nötigen Information zu.

Nachfolgend werden als Ausblick zwei Ansätze zur Bestimmung der konvexen Hülle in beliebigen Dimensionen vorgestellt.

## 4.2 Der Ansatz von CHAND/KAPUR

Bei der Methode, die von CHAND/KAPUR 1970 erstmals vorgestellt und von BHATTACHARYA 1982 analysiert wurde, bestimmt man zunächst eine Facette der konvexen Hülle einer $N$-elementigen Punktemenge $M = \{p_1, \ldots, p_N\}$ aus dem $\mathbb{R}^n$. Anschließend arbeitet man sich von einer Facette zur Benachbarten. Zur Veranschaulichung stellen wir uns vor, wir wollten ein 3-dimensionales Objekt in einen Papierbogen einwickeln. Hierzu muß zunächst eine Seite mit dem Papier überdeckt und anschließend nach und nach eine weitere benachbarte Seite mit dem Papierbogen abgedeckt werden, bis letztlich jede Seite einmal überdeckt worden ist. Daher wird diese Methode auch als "Geschenkeinwicklungsmethode" bezeichnet. Nach diesem Prinzip bestimmt der Algorithmus von AKL (vgl. 1.5) die konvexe Hülle einer Punktemenge in der Ebene.

Ein Algorithmus, der auf diese Weise die konvexe Hülle einer endlichen Anzahl von Punkten bestimmt, läuft nach folgendem Schema (vgl. [13], S. 131-136) ab:

1. Ermittle eine Facette $F$ der konvexen Hülle von $M$.
2. Schiebe alle Subfacetten $s_1, \ldots, s_l \subset F$, $l \in \mathbb{N}$ auf einen Stapel $S_1$.
3. Lege die Facette $F$ ebenfalls auf einen Stapel $S_2$.
4. Solange Elemente auf dem Stapel $S_2$ sind, wiederhole die Schritte (5)-(11).
5. Ziehe ein Element $E$ vom Stapel $S_2$ ab.
6. Ermittle die Subfacetten von $E$.
7. Für alle Subfacetten $s_1, \ldots, s_k$, $k \in \mathbb{N}$, die $E$ besitzt und die auf dem Stapel $S_1$ stehen, führe die Schritte (8)-(10) aus.
8. Ermittle die Facette $G$, die mit $E$ die Subfacette aus (7) gemeinsam hat.
9. Ziehe all die Subfacetten vom Stapel $S_1$ ab, die $G$ enthält, und schiebe alle Subfacetten der Nachbarfacette $G$, die noch nicht auf dem Stapel gespeichert waren, auf diesen Stapel.
10. Schiebe die Nachbarfacette $G$ auf den Stapel $S_2$.
11. Schreibe die Information über die Facette $E$ in ein File, oder gib diese direkt am Bildschirm aus.

Zur internen Darstellung der Facetten und Subfacetten benutzt man häufig sogenannte Flächengrafen. Hierauf werden wir im nächsten Abschnitt ausführlicher eingehen. Mit Hilfe von (9) wird der Gesamtaufwand erheblich reduziert, da unnötige Subfacetten nicht mehr auf den Stapel geschoben werden. Haben wir nämlich ausgehend von einer Facette $F$ über die Subfacette $s$ die Nachbarfacette $G$ bestimmt, so würde später die Facette $F$ in (8) als Nachbarfacette von $G$ ermittelt werden, wenn wir $G$ in (4) vom Stapel $S_2$ abziehen. Dies ist offensichtlich unnötig und würde zu einer Endlosschleife führen.

Nach ([13], Theorem 3.14) gilt nun für diesen Algorithmus nachfolgende Laufzeitabschätzung:

**Satz 4.2.1:** *Die konvexe Hülle einer N-elementigen Punktemenge aus dem $\mathbb{R}^n$ kann mit Hilfe des Algorithmus von CHAND/KAPUR im schlimmsten Fall mit einem Aufwand von*

$$O(N^{\lfloor d/2 \rfloor + 1}) + O(N^{\lfloor d/2 \rfloor} \log N)$$

*bestimmt werden.* ◇

## 4.3 Der Algorithmus von KALLAY

Lange Zeit war die "Geschenkeinwicklungsmethode" die einzige bekannte Methode zur Bestimmung konvexer Hüllen von endlichen vielen Punkten aus dem $\mathbb{R}^n$. Die Technik ist zwar in Hinblick auf ihre Laufzeit recht schnell, aber sie hat den Nachteil, daß sie nicht online arbeitet.

Nachfolgend stellen wir einen Ansatz von Kallay vor. Dieser arbeitet online. Die Grundlage für den Algorithmus bildet nachfolgendes Theorem, daß von McMullen und Shephard 1971 aufgestellt wurde:

**Satz 4.3.1:** *Sei $P$ ein Polytop, $p \in \mathbb{R}^n$ und $Q = co(P \cup p)$.*
*Dann kann jede Fläche von $Q$ folgendermaßen charakterisiert werden:*

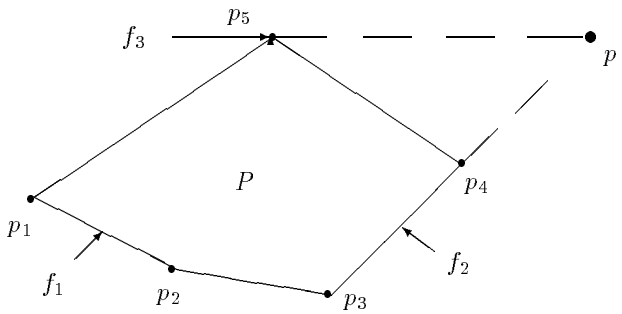
1. *Eine Fläche $f \in P$ ist genau dann auch eine Fläche von $Q$, wenn eine Facette $F \subset P$ existiert, so daß $f$ in $F$ enthalten ist, und $p$ diesseits von $F$ liegt.*
2. *Sei $f$ eine Fläche des Polytops $P$.*
   *Dann ist $g = co(f \cup p)$ genau dann eine Fläche von $Q$, falls*
   (a) *$p \in aff(f)$ oder*
   (b) *Es gibt mindestens eine Facette $F_1$ in $P$, die $f$ enthält, sodaß $p$ diesseits von ihr liegt. Darüber hinaus gibt es mindestens eine Facette $F_2$ in $P$, die ebenfalls $f$ enthält. $p$ liegt jedoch jenseits von $F_2$.*

◇

*Beweis.* Den Beweis findet der Leser in [11] auf Seite 113.

Zur Illustration des Satzes 4.3.1 betrachten wir folgendes zweidimensionales Beispiel:

**Beispiel 4.3.2:** Betrachte folgende Situation:



Die Punkte $p_1, \ldots, p_6$ aus der Ebene seien die Eckpunkte eines Polytops $P$. $p$ sei ein weiterer Punkt aus der Ebene. In diesem Fall folgt:

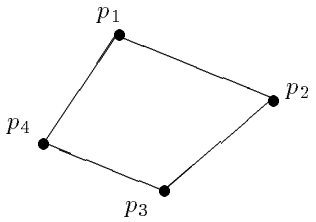
- Die Facette $f_1 := co(\{p_1, p_2\})$ erfüllt die Bedingung 4.3.1.(1).
- Die Facette $f_2 := co(\{p_3, p_4\})$ genügt der Bedingung 4.3.1.(2a).
- Die Subfacette $f_3 := \{p_5\}$ besitzt die Eigenschaften aus 4.3.1.(2b).

Folglich sind nach obigen Satz die Flächen $f_1, co(\{f_2, p\})$ und $co(\{f_1, p\})$ Flächen des Polytops $Q = co(P \cup p)$.

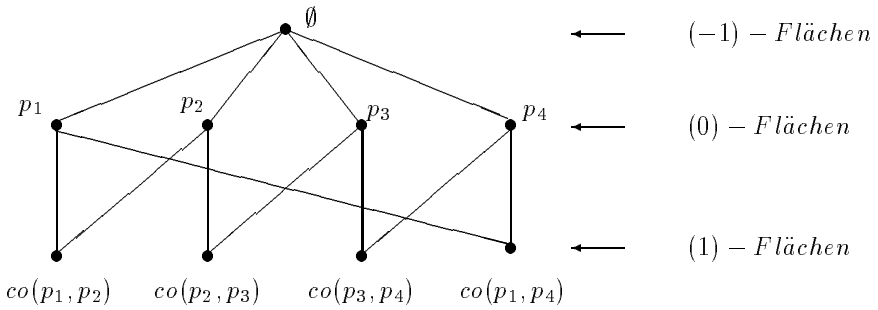
Die Daten der konvexen Hülle werden in Flächengraphen verwaltet. Diese sind wie folgt definiert:

**Definition 4.3.3:**
Sei $V$ eine Knotenmenge und $E$ eine Kantenmenge. So bezeichnet man das Paar $(V, E)$ als *Flächengraph der Dimension d*, falls jedes $v$ aus $V$ eine n-Fläche darstellt $(n = -1, 0, \ldots, d-1)$ und für jedes $e$ aus $E$ gilt: Ein zugehöriger Knoten repräsentiert eine (i-1)-Fläche $f$ und der andere Knoten eine i-Fläche $g$ mit $f \subset g$. $\diamond$

**Beispiel 4.3.4:** Betrachten wir die konvexe Hülle der Punkte $p_1, \ldots, p_4$ aus der Ebene:




Diese Hülle wird durch den folgenden Flächengraph repräsentiert:



Innerhalb eines Programms kann ein Flächengraph der Dimension d durch einfach verkettete Listen $L_{-1}, L_0, \ldots, L_{d-1}$ dargestellt werden. Diese Datenstruktur unterscheidet sich von den doppelt verketteten Listen (siehe 2.5.1) dadurch, daß es keinen Zeiger auf den Vorgängerknoten gibt. In der Liste $L_j$ wird die Information über die j-Flächen gespeichert. Jedes Element in $L_j$ entspricht dabei einer Fläche $f$. Zu jeder Fläche $f$ werden folgende Daten gespeichert:

- **BASE(f)**, die affine Basis der Fläche $f$, d. h. Information über die in $f$ enthaltenen 0-Flächen, die diese Basis bestimmen.
- **SUPER(f)**, Zeiger auf all die (j+1)-Flächen, die die Fläche $f$ enthalten.
- **SUB(f)**, Zeiger auf all die (j-1)-Flächen, die die Fläche $f$ enthält.
- **FACETT(f)**, Zeiger auf die Facetten, die die Fläche $f$ enthalten.

Der Algorithmus setzt nun voraus, daß die konvexe Hülle von $k \in \mathbb{N}$ Punkten $p_1, \ldots, p_k$ bekannt ist und in Form eines Flächengraphen $F_1$ hinterlegt ist. Bei Hinzunahme eines Punktes $p_{k+1}$ wird die konvexe Hülle der Punkte $p_1, \ldots, p_{k+1}$ berechnet und in einem Flächengraphen gespeichert. Hierfür erzeugt man zunächst einen zu $F_1$ isomorphen Flächengraphen $F_2$. Für diesen soll gelten:

$$F_2 := \{g | g = co(f \cup p_{k+1}); f \text{ ist eine Fläche aus } F_1\}$$

Anschließend eliminiert man aus $F_1$ und $F_2$ die Flächen, die in der konvexen Hülle der Punkte $p_1, \ldots, p_{k+1}$ nicht mehr vorkommen. danach wird der Flächengraph $F_2$ als Sohn des Wurzelknotens von $F_1$ angehängt. Um die Flächen zu bestimmen, die in $F_1$ und $F_2$ gelöscht werden können, überlegen wir uns Folgendes:

1. Im Flächengraph $F_1$ gibt es Flächen, die zwar die konvexe Hülle der Punkte $p_1, \ldots, p_k$ charakterisieren, jedoch nicht die konvexe Hülle der Punkte $p_1, \ldots, p_{k+1}$. Diese sind aus $F_1$ zu entfernen. Dazu betrachtet man die i-Flächen $f_j^i$ $(i, j \in \mathbb{N})$. Aufgrund von 4.3.1.(1) muß eine Fläche $f_j^i$ eliminiert werden, falls die affine Hülle jeder Facette, die die Fläche $f_j^i$ beinhaltet, den Raum so in zwei Halbräume teilt, daß der Punkt $p_{k+1}$ und einige Flächen der konvexen Hülle der Punkte $p_1, \ldots, p_k$ in verschiedenen Halbräumen liegen. (*)
   Wie leicht mit Hilfe eines kurzen Widerspruchsbeweises zu sehen ist, gilt nachfolgendes Lemma:

   **Lemma 4.3.5:** *Sei $P$ ein Polytop und $p \in \mathbb{R}^n$.*
   *Seien $f$ und $g$ Flächen aus $P$ mit $f \subset g$. Gilt*

   $$f \not\subset co(P \cup p),$$

*so folgt:*

$$g \not\subset co(P \cup p).$$

$\diamond$

Hiermit ist sichergestellt, daß zum einen in $F_1$ ganze Teilgraphen und zum anderen in $F_2$ einige Flächen gelöscht werden können.

2. Im Flächengraph $F_2$ gibt es Flächen, die nicht zur konvexen Hülle der Punkte $p_1, \ldots, p_{k+1}$ gehören. Diese müssen aus $F_2$ entfernt werden. Dazu betrachtet man die i-Flächen $f_j^i$ $(i, j \in \mathbb{N})$ von $F_1$. Wegen 4.3.1.(2) ist die Fläche $Co(f_j^i \cup p)$ zu eliminieren, falls die affine Hülle jeder Facette, die die Fläche $f_j^i$ beinhaltet, den Raum so in zwei Teilräume teilt, daß die Punkte $p_1, \ldots, p_{k+1}$ alle im gleichen Halbraum liegen. (**)
Um wiederum sicherzustellen, daß ganze Teilgraphen gelöscht werden können, benötigen wir das folgende Lemma, das mit Hilfe eines Widerspruchsbeweises leicht gezeigt werden kann:

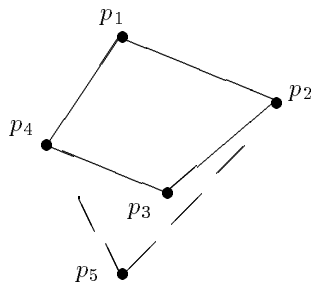
**Lemma 4.3.6:** *Sei $P$ ein Polytop und $p \in \mathbb{R}^n$.*
*Seien $f$ und $g$ Flächen aus $P$ mit $f \subset g$. Ist nun*

$$co(f \cup p) \not\subset co(P \cup p)$$

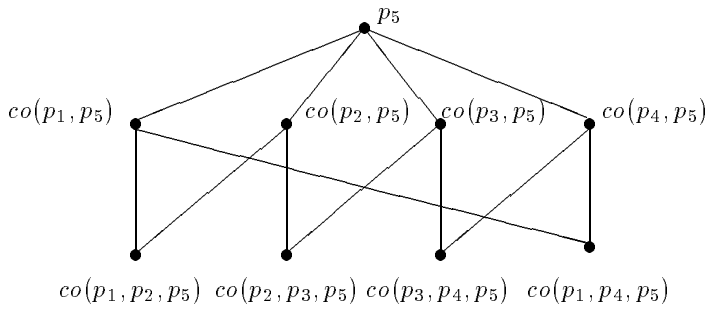
*erfüllt, so folgt*

$$co(g \cup p) \not\subset co(P \cup p).$$

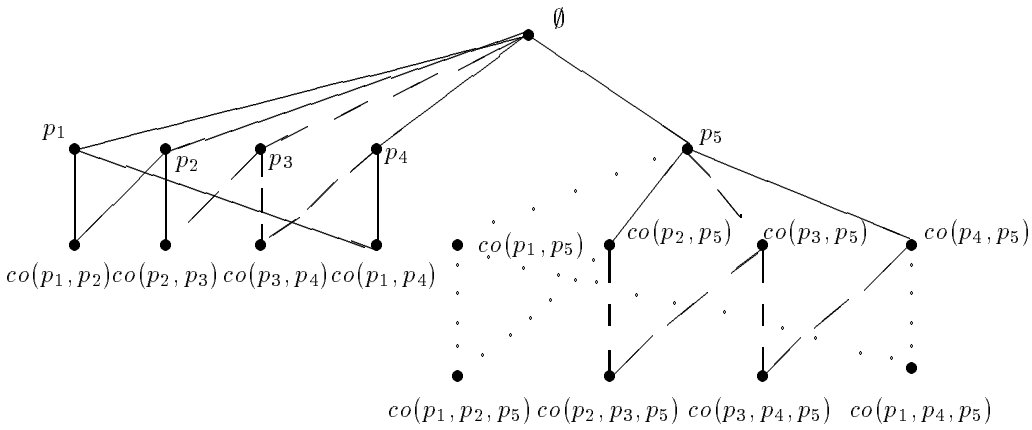
**Beispiel 4.3.7:** Wir betrachten nocheinmal die Punktemenge aus Beispiel 4.3.4 und ergänzen den Punkt $p_5$. Somit haben wir folgende Situation:




Wollen wir nun den Flächengraph bestimmen, der die konvexe Hülle der Punkte $p_1, \ldots, p_5$ darstellt, so ergibt sich aus dem Flächengraph $F_1$ aus Beispiel 4.3.4 zunächst durch Anwendung der Transformationsvorschrift der folgende Graph $F_2$:




Löscht man in den Graphen $F_1$ und $F_2$ die redundante Information, wie in (*) und (**) angegeben, und verbindet diese beiden wie oben erläutert, so resultiert der Graph $F$. Dieser repräsentiert die konvexe Hülle der Punkte $p_1, \ldots, p_5$.




Die gestrichelten Kanten sind aufgrund von (*) und die punktierten Kanten wegen (**) gelöscht worden.

Nach [13] gilt für den Ansatz von KALLAY folgende Laufzeitabschätzung:

**Satz 4.3.8:** *Wird die konvexe Hülle einer N-elementigen Punktemenge des $\mathbb{R}^d$ mit dem Algorithmus von Kallay bestimmt, so benötigt man dafür eine Zeitaufwand von $O(N^{\lfloor \frac{d}{2} \rfloor + 1})$.* ◇

**Bemerkung:** Will man mit obiger Technik die konvexe Hülle einer Punktemenge aus der Ebene bestimmen, so benötigt man hierfür einen Aufwand von $O(N^2)$. Somit ist diese Methode im zweidimensionalen Fall nicht optimal.

# Anhang A

# Das Programm GRAHAM

Möchte der Anwender die Eckpunkte einer konvexen Hülle einer endlichen Anzahl von Punkten mit Hilfe der vorliegenden Routinen bestimmen, so hat er Folgendes zu wissen:

1. **Benötigte Files:**
   - graha1.c
   - graham.h
   - numerik.c
2. **Realisierung:**
   Zur Lösung des Problems wurde der Algorithmus von Graham implementiert.
3. **Spezifikation:**
   Im File graha1.c befindet sich die Steuerfunktion "graham_abtastung". Sie hat folgende Gestalt:

   ```
   int graham_abtastung (dim,koord,anz_p) int dim;
                                          sPUNKT **koord;
                                          int anz_p;
   ```

   Der sPUNKT-Struktur liegt folgender Aufbau zugrunde:

   ```
   struct point {
                  double x;
                  double y;
                }
   ```

Sie geht aus dieser mittels der Typdefinition

$$typedef\ struct\ point\ sPUNKT;$$

hervor. Die Struktur sPUNKT dient zur internen Repräsentierung von Punkten aus der Ebene. Die Struktur wird im File *graham.h* definiert.

4. **Parameter:**

   dim INT
   Input: Dimension des Problems
   Output: unverändert

   koord sPUNKT ***
   Input: Adresse eines Feldes mit Elementen vom Datentyp sPUNKT, in denen die Punkte gespeichert sind, von denen man die Eckpunkte der zugehörigen konvexen Hülle bestimmen möchte.
   Output: Adresse eines Feldes mit Elementen vom Datentyp sPUNKT, in dem nur die Eckpunkte der konvexen Hülle aller zu Beginn gegebenen Punkte hinterlegt sind.

   anz_p INT *
   Input: Adresse eines Integerwertes, der angibt, wieviele Punkte in dem Feld, auf das *koord* zeigt, zu Beginn gespeichert sind.
   Output: Adresse eines Integerwertes, der die Anzahl der Eckpunkte der bestimmten konvexen Hülle angibt.

5. **Literatur:**
   Franco P.Preparata, Michael lan Samos:
   Computational Geometry - An Introduction; Springer Verlag; Berlin, Heidelberg, New York; 1985

6. **Erforderliche Unterprogramme:**
   keine

7. **Hilfsroutinen:**

   - void *allocate_liste* (daten,anz) DATES ***daten;
     int anz;
   - int *fuell_liste* (daten,koord,anz) DATES ***daten;
     sPUNKT **koord;
     int anz;

- int *best_in_punkt* (daten,anz) DATES **daten;
  int anz;
- void *trafo_punkte* (daten,anz) DATES **daten;
  int anz;
- void *vor_sort* (daten,anz) DATES **daten;
  int *anz;
- void *qsort* (daten,anz,width,fcomp) void * daten;
  int anz;
  int width;
  int (*fcomp)(void*,void*);
- void *gen_liste* (daten,anz) DATES **daten;
  int anz;
- int *get_one_edge* (daten,anz) DATES **daten;
  int anz;
- void *graham_scan* (daten,anz) DATES **daten;
  int anz;
- void *find_rand* (daten,anz) sPUNKT ***daten;
  int *anz;

Die Bedeutung der Funktionen sowie deren Parameter sind im Quellcode *graha1.c* dokumentiert. Der Struktur DATES liegt folgender Aufbau zugrunde:

```
struct dates {
              double x_pos;
              double y_pos;
              double range;
              double angle;
              int is_edge;
             }
```

Aus der Struktur dates erhält man vermöge

$$typedef\ struct\ dates\ DATES;$$

obige Struktur. Sie wird zur Realisierung einer einfach verketteten Liste sowie bei der Sortierung der Punkte aus der Ebene nach den Sortierkriterien "polarer Winkel und Entfernung zum Ursprung" benötigt.

8. **Beispiel:**
Wir möchten die Eckpunkte der konvexen Hülle all der Punkte bestimmen, die in dem File *daten.dat* zeilenweise gespeichert sind. Diese sollen in ein File mit Namen *result.dat* geschrieben werden. Der File *daten.dat*

hat folgenden Inhalt:

0.0 0.0
1.0 0.0
1.0 1.0
0.0 1.0
0.5 0.5

Das zugehörige c-File *myprog.c* sieht wie folgt aus:

```
#include "graham.h"
/* wird zur Definition der Struktur sPUNKT benoetigt */
main ()
    {
     sPUNKT **koord;
     int anzahl;
     lies_info2("daten.dat",&koord,&anzahl);
     graham_abtastung(2,&koord,&anzahl);
     print_points_to_file("result.dat",koord,anzahl);
     return;
    }
```

Befinden sich die Files *graha1.c* und *graham.h* im aktuellen Verzeichnis, so muß zunächst im Headerfile graham.h die Variable *GRAHAM* mit

```
#define GRAHAM
```

gesetzt werden. Mit Hilfe des UNIX-Utility MAKE und dem folgenden Makefile erhalten wir ein ausführbares Programm *myprog*:

```
myprog: graha1.o myprog.o numerik.o graham.h; \
 cc -Aa -D_HPUX_SOURCE -g -o myprog graha1.o myprog.o numerik.o -lm
graha1.o: graha1.c graham.h ; cc -Aa -D_HPUX_SOURCE -c -g graha1.c
myprog.o: myprog.c graham.h ; cc -Aa -D_HPUX_SOURCE -c -g myprog.c
numerik.o: numerik.c ; cc -Aa -D_HPUX_SOURCE -c -g numerik.c
```

Nachdem man das Programm *myprog* ausgeführt hat, sieht das File *result.dat* folgendermaßen aus:

```
   1.000000000000000    1.000000000000000
   0.000000000000000    1.000000000000000
   0.000000000000000    0.000000000000000
   1.000000000000000    0.000000000000000
   1.000000000000000    1.000000000000000
```

Der erste Punkt wurde am Ende des Files nocheinmal abgespeichert, damit man mit Hilfe von GNUPLOT die berechnete konvexe Hülle geschlossen zeichnen kann.

**Bemerkung:** In der Grahamschen Abtastung muß der Wert einer Determinanten ermittelt werden, um innere Punkte der konvexen Hülle bestimmen zu können. Diese kann entweder mit der Sarrus-Regel berechnet werden oder mittels einer QR-Zerlegung. Die zuerst angegebene Methode wird benutzt, falls die Variable $SARRUS\_REGEL$ im Headerfile mit

```
#define SARRUS_REGEL
```

definiert ist. Ist dies nicht der Fall, so wird eine QR-Zerlegung nach Householder angewandt, um die Determinante zu berechnen. ◇

## Anhang B

# Dokumentation zum Programm AUMANN

Möchte der Anwender Aumann-Integrale approximieren bzw. die Approximierungsverfahren analysieren, so muß er Folgendes wissen:

1. **Benötigte Files:**
   - graha1.c
   - numerik.c
   - aumann.c
   - func.bsp.c
   - func.h
   - graham.h
2. **Realisierung:**
   Zur Approximation von Aumannintegralen wurden Summenformeln verwendet, ähnlich wie bei gewöhnlichen Integralen. Es wurden folgende Ansätze implementiert:
   - Treppenregel
   - Trapezregel
   - Simpsonregel
3. **Spezifikation:**
   Im File *aumann.c* befinden sich die Steuerfunktionen *Aumann_integral* und *Aumann_integral2*. Sie haben folgende Gestalt:

   ```
   void Aumann_integral (bsp_nr,dim,a,b,F_p,f_dim,N,res,anz_p,verf_nr,konv_huelle)
                         int dim,f_dim,N,*anz_p,verf_nr;
   ```

```
/¿ int bsp_nr;
                    double a,b;
                    sPUNKT ***res;
                    void (*F_p);
                    int (*konv_huelle);
```

bzw.

```
void Aumann_integral2 (bsp_nr,dim,a,b,F_p,f_dim,N,res,anz_p
                       verf_nr,oldN,oldFeh,kosten_p,konv_huelle)
                       int bsp_nr,dim,f_dim,N,*anz_p,verf_nr,oldN;
                       double a,b,oldFeh;
                       sPUNKT ***res;
                       void (*F_p);
                       int (*konv_huelle);
                       double (*kosten_p)();
```

Die Funktion *Aumann_integral* berechnet lediglich die Ecken einer approximierenden Menge für ein Aumannintegral, während die Funktion *Aumann_integral*2 Information über die Konvergenzordnung, die Anzahl der Eckpunkte der approximierenden Menge sowie ein Maß für den benötigten Zeitaufwand ermittelt.

4. **Parameter:**
   Die Beschreibung der Parameter kann aus dem Quellcode von *aumann.c* entnommen werden.

5. **Literatur:**
   keine

6. **Erforderliche Unterprogramme:**
   keine

7. **Hilfsroutinen:**

   - void *alloc_points* (points,anz) sPUNKT ***points;
     int anz;
   - void *mengen_add* (m1,anz1,m2,anz2,m3_p,anz3_p) sPUNKT **m1,**m2;
     sPUNKT ***m3_p;
     int anz1,anz2;
     int *anz3_p;
   - int *scal_menge* (scal,points,anz) double scal;
     sPUNKT **points;
     int anz;

- void *free_points* (points,anz) sPUNKT **points;
  int anz;
- void *schr_res*2 (bsp_nr,nr,N,anz,auf,feh,ord) int bsp_nr,nr,N,anz;
  double auf,feh,ord;
- void *print_points*2 (bsp_nr,daten,anz,suff,vernr,N,ind) sPUNKT ** daten;
  int anz,ind,N,vernr;
  int bsp_nr;
  char suff[4];

8. **Allgemeine Hinweise:**
   Möchte der Anwender Aumannintegrale mit den hier implementieren Methoden approximieren, so muß er zuvor folgende Arbeitsschritte durchführen:

   (a) Die Variable $GRAHAM$ darf im Headerfile $graham.h$ nicht definiert sein.

   (b) Im HOME-Verzeichnis des Benutzers muß ein Verzeichnis mit Namen $Results$ angelegt werden. Befindet man sich im $HOME$-Verzeichnis, so kann man dieses Verzeichnis mit Hilfe des UNIX-Befehls

   ```
   mkdir Results
   ```

   einrichten. Dieser Arbeitsschritt muß nur einmal durchgeführt werden.

   (c) Der Anwender muß in einem File der Form

   $$bsp < bsp\_nr\{4\} > .ref$$

   Information über eine Referenzmenge zur Verfügung stellen. Dabei bezeichnet $< bsp\_nr\{4\} >$ die vierstellige Beispielnummer. Die Eckpunkte der Referenzlösung müssen zeilenweise in dem File eingetragen sein. Diese Information kann der Anwender entweder wie weiter unten beschrieben bereitstellen, oder er kann diese mit Hilfe eines anderen Programmes berechnen.

9. **Beispiel:**
   Wir wenden die implementierten Approximationsverfahren auf das Integral

   $$\int_a^b F(t)dt = \int_0^1 \begin{pmatrix} \sin(t) \\ \cos(t) \end{pmatrix} [-1,1]\, dt$$

   an. Es soll zu diesem Beispiel eine Tabelle erzeugt werden, die in Abhängigkeit von einem vorgegebenen Approximationsverfahren und einer Folge von Diskretisierungsparametern Auskunft über folgende Punkte geben soll:

- Anzahl der Eckpunkte der berechneten Näherungslösung
- ein Maß für den benötigten Rechenaufwand
- Genauigkeit der Näherungslösung; dazu wird der Hausdorffabstand zu einer Referenzlösung bestimmt
- Ordnung des gewählten Approximationsverfahrens

$F(t)$ wird offensichtlich durch die beiden Punkte $p_0(t) = (-1.0)\begin{pmatrix} \sin(t) \\ \cos(t) \end{pmatrix}$ und $p_1(t) = (1.0)\begin{pmatrix} \sin(t) \\ \cos(t) \end{pmatrix}$ bestimmt. Die Information wird in zwei Punkten vom Datentyp sPUNKT gespeichert. Dazu ergänzen wir im File *func.bsp.c*:

```
void F1(dim,t,pkt) double t;
                   int dim;
                   sPUNKT ** pkt;
    {
     pkt[0]->x=(-1.0)*(sin(t));
     pkt[0]->y=(-1.0)*(cos(t));
     pkt[1]->x=(1.0)*(sin(t));
     pkt[1]->y=(1.0)*(cos(t));
     return;
    }
```

Wir bezeichnen die Funktion mit F1, um anzudeuten, daß diese Funktion die erste Funktion ist, die wir betrachten.
In dem File *func.h* ergänzen wir den Prototypen der Funktion F1:

```
void F1 (int,double,sPUNKT**);
```

Im File *aumann.c* fügen wir in der *switch*-Anweisung der *main*-Funktion folgende Zeile ein:

```
case 1 : Aumann_integral2 (bsp_nr,2,0.0,2*M_PI,F1,2,N,&res,\
         &anz,ver,&oldN,&oldfeh,kosten,graham_abtastung);
```

Nachdem auf diese Weise die Files *aumann.c*, *func.h* und *func.bsp.c* geändert worden sind, erhalten wir mit Hilfe des MAKE-Utilities und folgendem Makefile ein ausführbares Programm:

```
aumann: graha1.o numerik.o aumann.o fkts.bsp.o func.h graham.h; \
        cc -Aa -D_HPUX_SOURCE -g -o aumann graha1.o aumann.o \
        numerik.o fkts.bsp.o -lm
graha1.o: numerik.c graha1.c graham.h; \
```

```
          cc -Aa -D_HPUX_SOURCE -c -g graha1.c
aumann.o: aumann.c graham.h func.h; \
          cc -Aa -D_HPUX_SOURCE -c -g aumann.c
fkts.bsp.o: fkts.bsp.c ; \
            cc -Aa -D_HPUX_SOURCE -c -g fkts.bsp.c
numerik.o: numerik.c; \
           cc -Aa -D_HPUX_SOURCE -c -g numerik.c
```

Im Anschluß daran können wir das ausführbare Programm *aumann* starten. Bei der Frage, welche Beispielnummer abgearbeitet werden soll, geben wir eine 1 ein. Als Verfahrensnummer wird eine Ziffer zwischen 1 und 3 eingegeben. Dabei haben die Ziffern folgende Bedeutung:

| **Ziffer** | **angewandte Approximationsmethode** |
|---|---|
| 01 | Treppenregel |
| 02 | Trapezregel |
| 03 | Simpsonregel |

Solange nun ein Diskretisierungsparameter größer Null eingegeben wird, wird für den jeweiligen Parameter eine Zeile in der Tabelle hinzugefügt. Diese hat für die Iterationsfolge (4,8,10,16) bei Verwendung der Simpsonregel folgendes Aussehen:

| Disk.Par. | Ecken | Aufwand | Fehler | Ordnung |
|---|---|---|---|---|
| 0004 | 0004 | 177.45 | 0.000570052178 | ———— |
| 0008 | 0004 | 354.89 | 0.000036410923 | ———— |
| 0010 | 0004 | 443.61 | 0.000014953226 | 3.988202789956 |
| 0016 | 0004 | 709.78 | 0.000002288212 | 3.992078978935 |

Die erstellte Tabelle wird nicht am Bildschirm ausgegeben. Die Daten werden im Verzeichnis

$HOME/Results

in ein File der Form

f<Beispielnummer{4}><Verfahrenskennziffer{2}>.inf

abgelegt. Die Ziffer in {}-Klammern gibt die Zahl der ausgegebenen Stellen an. Falls die Variable *TEX* im File *graham.h* definiert wird, wird im gleichen Pfad ein File mit Namen

f<Beispielnummer{4}><Verfahrenskennziffer{2}>.tex

angelegt. Die dort abgespeichert Information kann mittels des *input*-Befehls als Tabelle in ein LATEX-File eingebunden werden. Dazu muß im File *graham.h* im Abschnitt *notwendige Definitionen* die Variable TEX wie folgt definiert werden:

```
#define TEX
```

Möchte man die Tabelleninformation als ASCII-File abspeichern, so muß die entsprechende Zeile gelöscht oder auskommentiert werden.

Das Programm ermöglicht es dem Anwender, die berechneten Mengen zu visualisieren. Es werden mit Hilfe des Grafikpaketes GNUPLOT Bilder erzeugt, die eine Auskunft darüber geben, welche Mengen bei einer gewählten Mengensummationsformel bei der i-ten Mengensummation addiert werden und welche Menge hieraus resultiert. Es entstehen dabei Bilder, in denen drei Mengen dargestellt sind.

Nehmen wir an, wir hätten eine Summationsformel der Form

$$\sum_{i=0}^{N} c_i co(F(t_i)) \ (c_i \in \mathbb{R}_0^+, \ i = 0, \ldots, N)$$

gewählt. So stellt für $j \in \{1, \ldots, N\}$ eine Menge die Menge

$$A_j := \sum_{i=0}^{j-1} c_i co(F(t_i))$$

dar. Die Zweite die Menge

$$B_j := c_j co(F(t_j))$$

und die Dritte stellt das Ergebnis der Mengensummation

$$C_j := \sum_{i=1}^{j} c_i co(F(t_i))$$

dar. Wieviele Bilder erzeugt werden, legt der Anwender durch Definition der Variable MAX_PICT im File graham.h in der Sparte *notwendige Definitionen* fest. Sind der Wert der Variable MAX_PICT und der Diskretisierungparameter identisch, so wird für jeden Summationsschritt Information für ein Bild in einem File gespeichert. Beispielsweise wird mit

```
#define MAX_PICT 10
```

festgelegt, daß höchstens zehn Bildinformationsfiles angelegt werden sollen. Entspricht der Wert von $MAX\_PICT$ dem Diskretisierungsparameter, so wird zu jeder Mengensummation, die bei Anwendung einer Summationsformel stattfindet, ein entsprechendes File angelegt und das zugehörige Bild angezeigt. Die Eckpunkte der jeweils berechneten Mengen $A_j, B_j, C_j\,(j = 1, \ldots, N)$ werden im Verzeichnis

$HOME/Results

in Files der Form

kon<bsp_nr{4}>.<it_schr{4}.<verf_nr{2}>.<dis_par{4}>.<suf{3}>

abgespeichert. Dabei gelten die folgenden Abkürzungen:

| **Abkürzung** | **Bedeutung** |
|---|---|
| bsp_nr | Beispielnummer |
| it_schr | Iterationsschritt |
| verf_nr | Verfahrenskennziffer |
| dis_par | Diskretisierungsparameter |
| suf | Suffix |

Für die Suffixe gilt folgender Schlüssel:

| **Suffix** | **Dargestellte Menge** |
|---|---|
| old | A_j |
| add | B_j |
| ges | C_j |

Möchte der Anwender eine Folge von Bildern erzeugen und betrachten, so muß er im File *graham.h* noch die folgenden Änderungen durchführen:

(a) Die Variable SCHR_BILD muß mit

```
#define SCHR_BILD
```

gesetzt werden.

(b) Die Variable MAX_PICT muß wie oben angegeben definiert werden.

(c) Die Variable MAX_TIME gibt an, wie lange in Sekunden ein erzeugtes Bild von GNUPLOT auf dem Bildschirm angezeigt werden soll. Sie kann den eigenen Bedürfnissen entsprechend gesetzt werden. Zum Beispiel bewirkt die Definition

```
#define MAX_TIME 5
```

, daß jedes Bild für fünf Sekunden auf dem Bildschirm erscheint.

Nachdem die angegebenen Änderungen im Quellcode durchgeführt worden sind, muß mit Hilfe des MAKE-Utilities ein ausführbares Programm *aumann* wie oben angegeben erzeugt werden.

Wurde das Programm *aumann* aufgerufen und die abgefragten Werte eingegeben, so wird von diesem Programm das File *plot.hlp* erzeugt. Dieses wird im aktuellen Verzeichnis angelegt. Es beinhaltet zeilenweise *plot*-Anweisungen für das Programm *GNUPLOT*. Mit Hilfe der Hilfsdateien *plo*1.*hlp* und *plo*2.*hlp* sowie des UNIX-Befehls

```
cat plo1.hlp plot.hlp plo2.hlp >plo.gnu
```

kann der Anwender ein GNUPLOT-Skript *plo.gnu* erzeugen. Wird GNUPLOT zusammen mit diesem Skript aufgerufen, so werden die einzelnen Bilder am Bildschirm (oder in ein File) ausgegeben.

Nun kann das folgende Shell-Programm *showpic*1 benutzt werden, um zum einen mit Hilfe des Programms *aumann* die benötigten Bildinformationen in entsprechenden Files zu speichern und zum anderen die Bilder anzuzeigen:

```
aumann
cat plo1.hlp plot.hlp plo2.hlp >plo.gnu
gnuplot plo.gnu
```

Dafür müssen folgende Hilfsdateien im aktuellen Verzeichnis stehen:

- plo1.hlp
- plo2.hlp

Soll die Ausgabe der Bilder nicht auf den Bildschirm erfolgen, so muß im Headerfile die Variable *SCHR_BILD* und zusätzlich die Variable *SCHR_BILD_TEX* mit

```
#define SCHR_BILD
#define SCHR_BILD_TEX
```

gesetzt sein. Anschließend muß das Programm *aumann* — wie oben angegeben — durch Compilierung aktualisiert werden. Mit Hilfe des Shell-Skriptes *mkpspic* wird nun der gesamte Programmablauf gesteuert. Dieses sieht wie folgt aus:

```
aumann
cat plo3.hlp plot.hlp >plo.gnu
gnuplot plo.gnu
```

Um dieses Shellskript ablaufen zu lassen, müssen nachfolgende Hilfsdateien im aktuellen Verzeichnis stehen:

- plo3.hlp

Nun kann dieses Skript gestartet werden. Der Anwender wird daraufhin nach der Beispielnummer, der Verfahrenskennziffer und einmal nach einem Diskretisierungsparameter gefragt. Danach wird zunächst das File *plot.hlp* erzeugt und anschließend mittels des cat-Befehles ein GNUPLOT-Steuerskript erzeugt. Nach einem Aufruf des Programms GNUPLOT befinden sich im Verzeichnis

$$\$HOME/Results$$

EPS-Files mit Namen

$$\text{bild}<\text{bsp\_nr}\{4\}>.<\text{it\_schr}\{4\}>.<\text{verf\_nr}\{2\}>.<\text{dis\_par}\{4\}>.\text{ps}\ .$$

Diese können in LaTeX-Texte als EPS-Grafiken eingebunden werden.

Zusätzlich besteht die Möglichkeit, zu einer Folge von Diskretisierungsparametern jeweils die zugehörige berechnete approximierende Menge am Bildschirm anzeigen zu lassen. Zur Visualisierung der Mengen wird das Programm GNUPLOT verwendet. Hierzu muß im Headerfile die Variable $SCHR\_BILD\_APP$ mit

```
#define SCHR_BILD_APP
```

definiert werden. Nachdem das Programm *aumann* compiliert wurde, starten wir das Shellskript *showpic*1. Der Anwender wird nun wieder aufgefordert, die Nummer des zu betrachtenden Beispiels sowie die Kennziffer des ausgewählten Verfahrens einzugeben. Anschließend wird solange nach einem neuen Diskretisierungsparameter gefragt, bis hierfür ein Wert kleiner oder gleich Null eingegeben wird. Nach der letzten Eingabe wird am Bildschirm eine Folge von Bildern angezeigt, die zu einem gegebenen Diskretisierungsparameter $N$ die zugehörigen berechneten Näherungslösungen darstellen. Wieviel Zeit jeweils zwischen angezeigten Bildern verstreichen soll, kann der Anwender durch Definition der Variable $MAX\_TIME$ im Headerfile *graham.h* angeben. Die Definition

```
#define MAX_TIME 2
```

bewirkt beispielsweise, daß jedes Bild zwei Sekunden lang am Bildschirm angezeigt wird.

Möchte der Anwender die Bildinformation nicht am Bildschirm ausgeben, sondern ausdruckbare EPS-Files erzeugen, so kann er dies wie oben mit Hilfe des Shell-Skripts *mkpspic* erreichen: Hierfür müssen nachfolgende Hilfsdateien im aktuellen Verzeichnis stehen:

- plo3.hlp

Nun muß der Benutzer im Headerfile *graham.h* die Variable $SCHR\_BILD\_APP\_TEX$ mit

```
#define SCHR_BILD_APP_TEX
```

definieren. Nachdem der Anwender das ausführbare Programm *aumann* erzeugt hat, das Shellskript *mkpspic* gestartet hat und die abgefragten Werte eingegeben hat, werden im Verzeichnis

$HOME/Results

EPS-Grafikfiles angelegt. Diese können als EPS-Grafiken in LaTeX-Dokumente eingebunden werden. Die Dateien haben die Form

app<bsp_nr{4}>.<verf_nr{2}>.<dis_par{4}>.ps .

Möchte der Anwender lediglich eine Referenzlösung für ein Aumannintegral mit Hilfe eines der drei implementierten Methoden berechen lassen, so muß er im Headerfile *graham.h* die Variable $MK\_REF\_LOE$ mit

```
#define MK_REF_LOE
```

definieren. Nach der Compilierung der C-Files mit Hilfe des MAKE-Utilities und nach dem Aufruf des ausführbaren Programmes *graham* wird der Benutzer nach der Kennziffer der Methode, mit der eine Näherungslösung berechnet werden soll, sowie nach dem Diskretisierungsparameter gefragt. Anschließend wird die approximierende Menge bestimmt, die Koordinaten der Eckpunkte sortiert und zeilenweise in einem File der Form

kon<bsp_nr{4}>.<it_schr{4}.<verf_nr{2}>.<dis_par{4}>.ref

abgespeichert. Soll dieses File nicht angelegt werden, so ist die Definition

```
#define MK_REF_LOE
```

im Headerfile zu löschen.

## Anhang C

# Diskette mit dem Quellcode

# Literaturverzeichnis


[1] S.G. Akl ,G.T. Toussaint, "Efficient convex hull algorithms for pattern recognition applications", *Proc. 4th Int'l Joint Conf. on Pattern Recognition Kyoto, Japan, 1978, 483–487.*

[2] R. Baier, F. Lempio, "Approximating Reachable Sets by Extrapolation Methods". *In P. J. Laurent, A. Le Méhauté, and L. L. Schumaker, editors, Curves and Surfaces in Geometric Design, S. 9-18, Wellesley, Massachusetts, 1994. A K Peters.*

[3] R. Baier, "Mengenwertige Integration und die diskrete Approximation erreichbarer Mengen", *Dissertation, Mathematisches Institut Bayreuth, Deutschland, 1995.*

[4] J. L. Bentley, G. M. Faust, F. P. Preparata, "Approximation algorithms for convex hulls", *Comm. ACM 25, S. 64-68, 1982.*

[5] T. Bonnesen, W. Fenchel, "Theorie der konvexen Körper", *Ergebnisse der Mathematik und ihrer Grenzgebiete, Band 3, Heft1, Chelsea Publishing Company, Bronx-New York, 1934. Nachdruck: Chelsea Publishing Company, Bronx-New York, 1971.*

[6] D. R. Chand, S. S. Kapur, "An algorithm for convex polytopes", *JACM 17(1), S. 78-86,1970.*

[7] B. Chazelle, "An Optimal Convex Hull Algorithmus in Any Fixed Dimension", *Discrete & Computational Geometrie 10, Springer Verlag, New York, 1993.*

[8] K. L. Clarkson, P. W. Shor, "Applications of random sampling in computational geometry", *Discrete & Computational Geometrie 4, Springer Verlag, New York, S. 387-421, 1989.*

[9] M. Kallay, "Convex hull algorithms in higher dimensions", *unveröffentlichtes Skript, Dept. Mathematics, Univ. of Oklahoma, Norman, Oklahoma, 1981.*

[10] D.E. Knuth, "The Art of Computer Programming. Volume III: Sorting and Searching", *Addison-Wesley, Reading, Mass., 1973.*

[11] P. McMullen, G.C. Shephard, "Convex Polytopes and the Upper Bound Conjecture", *Cambridge University Press, Cambridge, England, 1971.*

[12] W. Müller, "Lineare Algebra", *Bayreuther Mathematische Schriften, Heft 24, Deutschland, 1987.*

[13] F. P. Preparata, M. Shamos, "Computational Geometry - An Introduction", *Springer-Verlag, New York, 1985.*

[14] F. P. Preparata, S. J. Hong, "Convex hulls of finite sets of points in two and three dimensions", *Comm. ACM 20(1977), S. 87-93.*

[15] A. Rosenfeld, "Picture Processing by Computers", *Academic Press, New York, 1969.*

[16] R. Seidel, "A convex hull algorithm optimal for points in even dimensions", *Technical Report 81-84, University of British Columbia, 1981.*

[17] R. Seidel, "Constructing higher-dimensional convex hulls at logarithmic cost per face", *Proc. 18th Annual ACM Symp. on Theory of Computing, S. 404-413, 1986.*

[18] J. Stoer, "Numerische Mathematik", *5. Auflage, Springer-Verlag, Berlin, Deutschland, 1989.*

[19] J. Stoer, C. Witzgall, "Convexity and Optimization in Finite Dimension I", *Springer-Verlag, New York, 1970.*

[20] V. M. Veliov, "Second order discrete approximations to linear differential inclusions", *SIAM Journal on Numerical Analysis, 29(2): 439-451, 1992.*

[21] V. M. Veliov, "Discrete approximations of integrals of multivalued mapping", *Comptes rendus de l' Académie bulgare des Sciences, 42(12): 51-54, 1989.*

[22] N. Wirth, "Algorithmen und Datenstrukturen", *Teubner, Deutschland, 1983.*

# Erklärung

Ich versichere hiermit, daß ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Hilfsmittel angefertigt habe.

Bayreuth, 30. Oktober 1995