Automatische Bestimmung der konvexen Hülle einer endlichen Punktmenge in der Ebene

Martin Rheinländer
Fachbereich Mathematik & Statistik, Universität Konstanz

Seminarvortrag
18. Mai 2005

Was ist eine konvexe Menge bzw. konvexe Hülle?

Definition 1: Eine Teilmenge $K \subset \mathbb{R}^n$ heißt *konvex*, wenn für alle Punkte $\mathbf{x}, \mathbf{y} \in K$ die gemeinsame Verbindungslinie vollständig in K enthalten ist, also:

$$\mathbf{x}, \mathbf{y} \in K \quad \text{und} \quad \theta \in [0,1] \qquad \Rightarrow \qquad (1-\theta)\mathbf{x} + \theta\mathbf{y} \in K.$$

Definition 2: Es seien $\lambda_1, \ldots, \lambda_n$ nichtnegative Zahlen mit $\lambda_1 + \ldots + \lambda_n = 1$. Dann heißt $\lambda_1\mathbf{x}_1 + \ldots\lambda_n\mathbf{x}_n$ *Konvexkombination* von $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^n$.

Satz 1: Ist $K \subset \mathbb{R}^n$ konvex, so enthält K jede Konvexkombination seiner Elemente.
Beweis per vollständiger Induktion über Zahl der konvexkombinierten Elemente.

Eigenschaften konvexer Mengen:

Stabiltät bzgl. Durchnitts- und Summenbildung,d.h. K, L konvex $\Rightarrow K \cap L, K + L$ konvex.
Projektionseigenschaft vollständiger, konvexer Mengen in Hilberträumen, d.h. Existenz genau eines Elemets mit minimaler Norm. Beachte: U konvex $\Rightarrow$ $\mathrm{Cl(U)}$ (Abschluß) konvex,

Definition 3: Die *konvexe Hülle* $\mathscr{C}(U)$ einer Teilmenge $U \subset \mathbb{R}^n$ ist die kleinste konvexe Menge, welche U enthält. Genauer entspricht die konvexe Hülle dem Durchschnitt aller konvexer Teilmengen des $\mathbb{R}^n$, welche U enthalten. (Beachte: U offen $\Rightarrow \mathscr{C}(U)$ offen.)

Bemerkung 1: Konvexe Hülle $\mathscr{C}(U) = \{$Menge aller Konvexkombinationen der Elemente aus $U\}$
Satz(Krein-Milman) 2: In einem normierten Raum ist jede kompakte, konvexe Menge die abgeschlossene, konvexe Hülle ihrer Extremalpunkte.

Konvexe Hülle einer endlichen Punktmenge

Anschauung:

- Endliche Punktwolke $=$ herausragende Nägelköpfe in einem Brett.
- Um die Nägelköpfe gelegtes Gummiband, minimiert seine Länge, indem es sich so weit wie möglich zusammenzieht.
- Konvexe Hülle $=$ vom Gummiband umschlossene Fläche.

motiviert folgende, alternative Definition:

Satz 3: Die konvexe Hülle einer Menge von endlich vielen Punkten in der Ebene $\mathcal{P} \subset \mathbb{R}^2$ ist durch das eindeutige, konvexe Polygon P gegeben, dessen Eckpunkte sämtlich in $\mathcal{P}$ enthalten sind und welches zugleich alle Punkte von umschließt, d.h. $\mathcal{P} \subset P$.

Beweis: Es sei P ein konvexes Polygon mit den verlangten Eigenschaften: $\mathcal{P} \subset P$ und jede Seite von P ist durch die Verbindungslinie eines Punktepaares aus $\mathcal{P}$ gegeben.
Annahme: $P \neq \mathscr{C}(\mathcal{P}) \Rightarrow \exists\, \mathbf{q} \in P \backslash \mathscr{C}(\mathcal{P})$. Wegen Vorausset.: $\partial P \subset \mathscr{C}(\mathcal{P}) \Rightarrow \mathbf{q} \in \mathrm{Int}(P)$.
Lege beliebige Gerade G durch $\mathbf{q}$. Da G, P konvex und abgeschlossen $\Rightarrow$ $G \cap P$ konvex und abgeschlossen. Außerdem besteht $G \cap P$ aus mehr als einem Punkt, da $\mathbf{q} \in \mathrm{Int}(P)$.
$\Rightarrow G \cap \partial P = \{\mathbf{s}_1 \neq \mathbf{s}_2\}$. Nun $\mathbf{s}_1, \mathbf{s}_2 \in \partial P \subset \mathscr{C}(\mathcal{P})$ aber: $\mathscr{C}(\mathcal{P}) \not\ni \mathbf{q} \in \overline{\mathbf{s}_1\mathbf{s}_2} \subset \mathscr{C}(\mathcal{P})$.
Widerspruch, also $P = \mathscr{C}(\mathcal{P})$.
Eindeutigkeit der konvexen Hülle $\Rightarrow$ Eindeutigkeit von P. Existenz von P ergibt sich konstruktiv. Siehe Algorithmen auf den folgenden Seiten. ■

Bestimmung der Konvexen Hülle: intuitive Ansätze

− *Ursprüngliche* Definition (Def.1& Bem.3) der konvexen Hülle: *nicht praktikabel* $\Rightarrow$ *unbrauchbar*

\+ nutze Beobachtung des vorangehenden Satzes; beachte: Polygon ist ein "endliches" Objekt

konvexe Hülle = Polygon, dessen Seiten ausgewählten Verbindungslinien entsprechen

$\Rightarrow$ **erster, intuitiver Ansatz:**

- Durchlaufe alle Punktepaare $(\mathbf{p}, \mathbf{q}) \in \mathcal{P} \times \mathcal{P}$ mit $\mathbf{p} \neq \mathbf{q}$ und überprüfe, ob $\overline{\mathbf{pq}}$ eine Kante von $\mathscr{C}(\mathcal{P})$ darstellt.
- *Entscheidungskriterium:* Die Gerade durch $\mathbf{p}, \mathbf{q}$ muß die Ebene so in zwei Halbebenen zerschneiden, daß sich $\mathcal{P}$ genau auf einer Seite befindet.

weitere Ansätze:

- Die konvexe Hülle ist dadurch charakterisiert, daß sie das Polygon mit dem kleinsten Umfang darstellt, dessen Eckpunkte allesamt zu $\mathcal{P}$ gehören und zugleich $\mathcal{P} \subset P$.
- Ein sehr konstruktives Verfahren besteht in dem folgenden Vorgehen: Angenommen die konvexe Hülle von $\mathcal{P}_k := \{\mathbf{p}_1, \ldots, \mathbf{p}_k\}$ mit $k < \#\mathcal{P}$ sei bereits konstruiert. Die konvexe Hülle von $\mathcal{P}_{k+1}$ ergibt sich dadurch, daß man $\mathbf{p}_{k+1}$ mit den beiden Eckpunkten von $\mathscr{C}(\mathcal{P}_k)$ verbindet, welche jeweils über einen von $\mathbf{p}_{k+1}$ aus "sichtbaren" und "unsichtbaren" Nachbarn verfügen, und die "sichtbaren" Kanten streicht. Der Algorithmus startet mit dem Dreieck, das durch $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$ gegeben ist. (Achtung im Entartungsfall: linienförmiges Dreieck)

Algebraische Umsetzung des Entscheidungskriteriums

Gegeben: gerichtete Gerade g von $\mathbf{p} \in \mathbb{R}^2$ nach $\mathbf{q} \in \mathbb{R}^2$. *Gesucht:* Lage von $\mathbf{r} \in \mathbb{R}^2$ bzgl. g.
Lemma: Die Lage des Punktes $\mathbf{r}$ läßt sicht mittels des Vorzeichens der Determinante D bestimmen, genauer:

$$\operatorname{sgn} \underbrace{\begin{vmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{vmatrix}}_{:=D=D(\mathbf{p},\mathbf{q},\mathbf{r})} = \begin{cases} 1: & \mathbf{r} \text{ liegt in der } \textit{linken} \text{ Halbebene} \\ 0: & \mathbf{r} \text{ liegt auf der Geraden durch } \mathbf{p}, \mathbf{q} \\ -1: & \mathbf{r} \text{ liegt in der } \textit{rechten} \text{ Halbebene} \end{cases}$$

Beweis: (3.Zeile - 1. Zeile), (2. Zeile - 1. Zeile). Entwickeln nach der 1. Spalte:

$$D = \begin{vmatrix} q_x - p_x & q_y - p_y \\ r_x - p_x & r_y - p_y \end{vmatrix} = \begin{pmatrix} -(q_y - p_y) \\ q_x - p_x \end{pmatrix} \cdot \begin{pmatrix} r_x - p_x \\ r_y - p_y \end{pmatrix} = R_{\pi/2}(\mathbf{q} - \mathbf{p}) \cdot (\mathbf{r} - \mathbf{p})$$

$R_{\pi/2}(\mathbf{q} - \mathbf{p})$ ergibt sich aus dem von $\mathbf{p}$ nach $\mathbf{q}$ weisenden Verbindungsvektor $\mathbf{q} - \mathbf{p}$ durch Drehung um 90^o gegen den Uhrzeigersinn (mathematisch positiver Sinn). Somit ist $R_{\pi/2}(\mathbf{q} - \mathbf{p})$ kollinear zum Normalenvektor der Geraden und zeigt in die *linke* Halbebene.
Es sei $\phi := \angle\big(R_{\pi/2}(\mathbf{q} - \mathbf{p}), \mathbf{q} - \mathbf{p}\big)$ mit $0 \le \phi < \pi$.

$$\operatorname{sgn} D = \operatorname{sgn} \cos(\phi) = \begin{cases} 1 & : & \phi < \frac{\pi}{2} & \Leftrightarrow & \mathbf{r} \text{ liegt in der linken Halbebene} \\ 0 & : & \phi = \frac{\pi}{2} & \Leftrightarrow & \mathbf{r} - \mathbf{p} \text{ kollinear zu } \mathbf{q} - \mathbf{p}, \text{ d.h. } \mathbf{r} \in g \\ -1 & : & \phi > \frac{\pi}{2} & \Leftrightarrow & \mathbf{r} \text{ liegt in der rechten Halbebene} \end{cases} \quad \blacksquare$$

Ein erster Algorithmus

Input: Liste der Punkte $\mathbf{p}_1, \ldots, \mathbf{p}_n \in \mathbb{R}^2$
Output: Liste der Randpunkte der konvexen Hülle gegen den Uhrzeigersinn durchlaufen (Inneres links)

```
l = 0; //counts #edges
for i := 1 : n,
  for j := 1 : n,
    if(j ≠ i)
      valid := true;
      for k := 1 : n,
        if(k ≠ i & k ≠ j)
          if(isAtRight(p_i, p_j, p_k)) valid := false; break; end;
        end; //if
      end; //k
      if(valid = true) l := l + 1; edge(l).foot := p_i; edge(l).head := p_j; end;
    end; //if
  end; //j
end; //i
```

// edge-list is to be sorted, such that edges are passed counterclockwise and domain is at l.h.s.

$$isAtRight(\mathbf{p}_i, \mathbf{p}_j, \mathbf{p}_k) = \begin{cases} \textbf{true}: & \text{falls} \quad D(\mathbf{p}_i, \mathbf{p}_j, \mathbf{p}_k) > 0 \\ \textbf{false}: & \text{falls} \quad D(\mathbf{p}_i, \mathbf{p}_j, \mathbf{p}_k) < 0 \end{cases}$$

Berücksichtigung des Sonderfalles, daß mehrere Punkte auf einer Linie liegen:

$$isAtRight(\mathbf{p}_i, \mathbf{p}_j, \mathbf{p}_k) = \begin{cases} \ldots & \text{(siehe oben)} \\ \textbf{false}: & \text{falls} \quad D(\mathbf{p}_i, \mathbf{p}_j, \mathbf{p}_k) = 0 \quad \wedge \quad \mathbf{p}_k \notin \overline{\mathbf{p}_i, \mathbf{p}_j} \\ \textbf{true}: & \text{falls} \quad D(\mathbf{p}_i, \mathbf{p}_j, \mathbf{p}_k) = 0 \quad \wedge \quad \mathbf{p}_k \in \overline{\mathbf{p}_i, \mathbf{p}_j} \end{cases}$$

Asymptotische Abschätzung des Aufwands bzgl. der Anzahl von Punkten $n = \#\mathcal{P}$

- $n(n-1) = n^2 - n$ *gerichtete* Verbindungslinien $\overrightarrow{\mathbf{p}_i\mathbf{p}_j}$
- $(n-2)$-maliges Auswerten der Funktion $isAtRight$ pro gerichtete Verbindungslinie (ohne **break**)
- Ordnen der Kanten-Liste mit m Kanten: schlimmstenfalls $m + (m-1) + \ldots + 1 = \frac{(m+1)m}{2}$ Durchläufe (bei lineare Suche). Da $m \leq n$, verhält sich der Aufwand für diesen Teilschritt wie $\mathcal{O}(n^2)$, was sich asymptotisch nicht bemerkbar macht.

$$\Rightarrow \textbf{Geasamtaufwand} \propto \mathcal{O}(n^3)$$

Bestimmung der konvexen Hülle: inkrementeller Ansatz

Lineare Ordnung der Punkte im $\mathbb{R}^2$: $\mathbf{q} > \mathbf{p} :\Leftrightarrow \begin{cases} q_x > p_x \\ q_x = p_x \quad \wedge \quad q_y > p_y \end{cases}$

- lexikographisches Sortieren der Punkte: angenommen $\mathbf{p}_1 < \ldots < \mathbf{p}_n$.
  beachte: $\mathbf{p}_1$ und $\mathbf{p}_n$ gehören zur konvexen Hülle (Extremalpunkte)
- inkrementeller (schrittweiser) Ansatz: bestimme zunächst konvexe Hülle der ersten k Punkte, füge dann einen weiteren Punkt hinzu und passe die konvexe Hülle entsprechend an.
- bestimme getrennt:
  - *obere konvexe Hülle:* von $\mathbf{p}_1$ nach $\mathbf{p}_n \rightarrow$ Rechtskurve (im Uhrzeiger)
  - *untere konvexe Hülle:* von $\mathbf{p}_1$ nach $\mathbf{p}_n \rightarrow$ Linkskurve (gegen Uhrzeiger)

Algorithmus zur Bestimmung der unteren konvexen Hülle - obere konvexe Hülle analog
Input : lexikographisch sortiertes Array der Punkte aus $\mathcal{P}$: $\mathbf{p}_1 < \ldots < \mathbf{p}_n$

```
lower_hull.append(p_1); lower_hull.append(p_2); //initializes list of hull nodes
for i := 3 : n,
  lower_hull.append(p_i);
  while(more than 2 nodes in lower_hull)
    if (isStrictRightTurn(last_but_two, last_but_one, last of lower_hull))
      lower_hull.erase_last_but_one; else break; end;
  end; /* while */   end; /* for */
```

$$isStrictRightTurn(\mathbf{p},\mathbf{q},\mathbf{r}) = \begin{cases} \textbf{true}: & \text{falls } D(\mathbf{p},\mathbf{q},\mathbf{r}) < 0 \quad (\mathbf{r} \text{ liegt rechts von/auf } \overrightarrow{\mathbf{pq}}) \\ \textbf{false}: & \text{falls } D(\mathbf{p},\mathbf{q},\mathbf{r}) \geq 0 \quad (\mathbf{r} \text{ liegt links von } \overrightarrow{\mathbf{pq}}) \end{cases}$$

Beachte: Der Polygonzug, welcher in der Liste $lower\_hull$ durch den Algorithmus sukzessive aufgebaut wird, erfüllt die beiden folgenden Eigenschaften:

i) Es gibt keine Punkte in $\mathcal{P}$, welche direkt unter dem Polygonzug liegen.

ii) Der Polygonzug vollführt keinen Rechtsknick (entweder geradeaus oder Rechtsknick).

Beide Eigenschaften werden vom Algorithmus gleichzeitig erzwungen. Denkbar wäre auch ein Vorgehen, daß erst einen Polygonzug mit der Eigenschaft i) ermittelt und dann ii) durch Korrekturen erzwingt.
Völlig entsprechend kann man einen Polygonzug für die obere konvexe Hülle konstruieren. Soll der Rand der konvexen Hülle gegen den Uhrzeigersinn durchlaufen werden, ist $upper\_hull$ in "umgedrehter" Reihenfolge an $lower\_hull$ zu hängen.

Asymptotische Abschätzung des Aufwands bzgl. der Anzahl von Punkten $n = \#\mathcal{P}$
Beim Durchlaufen der **for**-Schleife wird jeder Punkt zunächst in die Liste $lower\_hull$ geschrieben $\Rightarrow$ n-maliges Aufrufen von $lower\_hull.append$. Die Funktion $isStrictRightTurn$ wird mindestens einmal in jedem **for**-Schleifendurchgang aufgerufen. Desweiteren wird sie wiederholt aufgerufen, wenn Punkte aus der Liste $lower\_hull$ gestrichen wurden. Da jeder Punkt nur einmal in die Liste eingetragen wird, kann er höchstens ein einziges Mal gestrichen werden $\Rightarrow$ $\mathcal{O}(n)$ Aufrufe der Funktion $isStrictRightTurn$.
$\Rightarrow$ *Die Bestimmung der konvexen Hülle einer* sortierten *Punktliste ist in linearer Zeit möglich.*

Vergleich der beiden Algorithmen

Aufwand

1. Verfahren: $\mathcal{O}(n^3)$
2. Verfahren: $\mathcal{O}\big(n\log(n)\big)$ (Da das Sortieren im wesentlichen den Aufwand bestimmt!)

Robustheit = Stabilität bzgl. kleiner Rechenfehler: Ein Algorithmus heißt *robust*, wenn er trotz endlicher Rechengenauigkeit imstande ist, sinnvolle Ergebnisse zu liefern. Diese Eigenschaft ist von großer praktischer Bedeutung, da die Gleitkomma-Arithmetik (floating-point), welche standardmäßig auf jedem Rechner zur Verfügung steht, nur endlich viele Nachkomma-Stellen berücksichtigen kann und somit bei den meisten Operationen Rundungsfehler auftreten. Rechenungenauigkeiten machen sich insbesondere bei Entartungen (Sonderfällen/degeneracies) bemerkbar; hier z.B. mehrere Punkte, welche (fast) auf einer Geraden liegen oder doppeltes Vorkommen eines Punktes.

Entartung: (Beinahe)-Kollinearität mehrerer Punkte
Verfahren 1: nicht robust. Rundungsfehler können dazu führen, daß von einem Eckpunkt der konvexen Hüle mehr als zwei oder gar keine Kanten ausgehen. Damit ist das Ergebnis kein (einfach) geschlossener Polygonzug (closed polygonal chain). $\Rightarrow$ Crash !
Verfahren 2: robust. Es ist z.B. denkbar, daß durch Rundungsfehler ein minimaler Rechtsknick als ein geringfügiger Linksknick interpretiert wird bzw. gar nicht als Knick erkannt wird. In einem solchen Fall, kann ein Punkt außerhalb der ermittelten konvexen Hülle liegen. Der Algorithmus liefert aber in jedem Fall einen geschlossenen Polygonzug als Ergebnis!

Umsetzung (z.B. in C++)

Ingredienzien des Programms:

- Rechnen mit Punkten bzw. Vektoren aus dem $\mathbb{R}^2$
- Realisation mittels einer eigenständigen Klasse (z.B. `Real2, MWPoint`)
  - Grundrechenarten (Addieren, Subtrahieren, Skalarprodukt)
  - Zuweisung, Ausgabefunktion, *Vergleichsoperator*
  - Determinante $\Rightarrow isAtRight, isStrictLeftTurn, isStrictRightTurn$
- Liste als Container Template-Klasse
  beachte: Liste kann nur Elemente des gleichen Typs aufnehemen.
- Funktionalität der Liste
  - Anhängen, Einfügen und Löschen von Elementen
  - Ausgabefunktion (zum Debuggen bzw. zur Anzeige des Ergebnisses)
    setzt Ausgabefunktion für den Elementetyp voraus
  - Sortierfunktion (setzt einen Vergleichsoperator für den Elementetyp voraus!)
- graphische Ein- und Ausgabe (Widget-Bibliothek QT)

Literaturangabe/Vortragsgrundlage: erstes Kapitel des Springer-Lehrbuchs *Computational Geometry, Algorithms and Applications* von *M.de Berg, M.van Kreveld, M. Overmars* und *O.Schwarzkopf*.