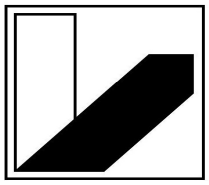

UNIVERSITÄT
BAYREUTH

FAKULTÄT FÜR MATHEMATIK UND PHYSIK
MATHEMATISCHES INSTITUT

Mengenorientierte optimale Steuerung und Fehlerschätzung

Diplomarbeit

von

Marcus von Lossow

Datum: 16. November 2005

Aufgabenstellung / Betreuung:
Prof. Dr. Lars Grüne

Danksagung

An dieser Stelle möchte ich mich bei Herrn Professor Grüne für die ausgezeichnete Betreuung während der Arbeit und das Heranführen an diesen Themenbereich durch seine Vorlesungen und Seminare bedanken.
Mein Dank gilt auch meinen Eltern und meiner Freundin für die Unterstützung während des Studiums.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Optimale Steuerungsprobleme spielen bei technischen und wirtschaftlichen Aufgabenstellungen eine große Rolle. Durch die Entwicklung immer leistungsfähigerer Computer können heutzutage einige Steuerungs- bzw. Regelungsprobleme aus der Realität am Rechner gelöst werden. Dennoch stoßen heutige Rechner mit den bekannten Algorithmen bei höherdimensionalen Problemen, die in der Realität häufig vorkommen, an ihre Grenzen. Deshalb ist es wichtig, die Algorithmen zur optimalen Steuerung zu verbessern und neue Algorithmen zu entwerfen.

In dieser Arbeit wird ein graphentheoretischer Ansatz zur mengenorientierten optimalen Steuerung untersucht. Es werden hier diskrete nichtdiskontierte Steuerungsprobleme über einem unendlichen Zeithorizont betrachtet.
Ein Beispiel für ein solches Problem ist ein Transportproblem. Ein Gegenstand soll auf kürzestem Weg vom Startort zum Zielort transportiert werden. Dies führt zu einem Kürzeste-Wege-Problem, das mit graphentheoretischen Algorithmen, z.B. mit dem Dijkstra-Algorithmus, effizient gelöst werden kann. In dieser Arbeit wird dargelegt, wie man auch allgemeine Steuerungsprobleme als ein Kürzestes-Wege-Problem darstellen kann.
Räumliche Diskretisierungen des Zustandsraumes werden oft zur Lösung dieser Steuerungsprobleme verwendet (siehe z.B. [7]). Bei Diskretisierung des Raumes in n Zellen beträgt der Aufwand bei Verwendung der Prinzipien der dynamischem Programmierung $O(n^2)$. Eine Einführung in die dynamischen Programmierung ist in [1] zu finden. Die Lösung des Kürzeste-Wege-Problems mit n Knoten und e Kanten erfordert einen Aufwand von $O(n \log(n) + e)$ (Zum Beweis siehe Abschnitt 8.5.1 von [13]). Bei dem hier betrachteten Problem ist die Zahl der Kanten etwa proportional zu der Zahl der Zellen, somit liegt der graphentheoretische Algorithmus in einer besseren Komplexitätsklasse als der Ansatz mit dynamischer Programmierung.

Im Kapitel 2 werden grundlegende Begriffe der Graphentheorie und weitere Grundbegriffe,

die bei der Analyse von Kontrollsystemen benötigt werden, eingeführt.
Die hier betrachteten Kontrollsysteme werden mit ihren elementaren Eigenschaften im Kapitel 3 behandelt.
Im Kapitel 4 wird der graphentheoretische Ansatz dargestellt und die Konvergenz der approximierten Wertefunktion bewiesen. Dabei wird der Darstellung in [10] gefolgt.
Im Kapitel 5 wird eine Feedback-Regelung eingeführt, um das System zu steuern, und die Güte der erhaltenen Trajektorien betrachtet. Des Weiteren wird ein Fehlerschätzer konstruiert, der eine adaptive Verfeinerungsstrategie erlaubt. Dieses Kapitel orientiert sich an [8]. Weitere Ergebnisse zu dem Fehlerschätzer beruhen auf eigenen Untersuchungen.
Im nächsten Kapitel wird auf die Implementierung dieses Ansatzes eingegangen.
Im Kapitel 7 werden anhand zweier Beispiele numerische Ergebnisse präsentiert.
Schließlich wird im Kapitel 8 ein Ausblick auf mögliche Verbesserungen des Ansatzes und auf Verallgemeinerungen gegeben.

Kapitel 2

Grundbegriffe

2.1 Graphentheorie

Eine Einführung in die Graphentheorie findet sich in [2]. Zunächst folgen einige grundlegende Definitionen aus der Graphentheorie.

Definition 2.1. *Ein* **gerichteter Graph** $G = (V(G), E(G))$ *ist ein 2-Tupel bestehend aus* $V(G)$*, der* **Knotenmenge***, einer nichtleeren Menge, und* $E(G) \subset V(G) \times V(G)$*, der* **Kantenmenge***. Die Elemente von* $V(G)$ *heißen* **Knoten von G***, die Elemente von* $E(G)$ *heißen* **Kanten von G***.*

Definition 2.2. *Ein gerichteter Graph* $G = (V(G), E(G))$ *heißt* **endlich***, falls er nur aus endlich vielen Knoten besteht.*

Definition 2.3. *Ein* **bewerteter Graph** *ist ein gerichteter Graph* G*, in dem jeder Kante* e *eine reelle Zahl* $w(e)$ *zugeordnet wird, die als* **Bewertung** *oder Länge von* e *bezeichnet wird.*

Annahme 2.4. *Im Folgenden werden nur endliche, bewertete Graphen mit* $w(e) \geq 0 \ \ \forall e \in E(G)$ *betrachtet.*

Ist $p = (s = v_0, v_1, \ldots, v_k = t)$ ein Weg von einem Knoten s über die Knoten $v_1, \ldots, v_{k-1}$ zu einem Knoten t, so wird die Länge von p als Summe der Bewertungen seiner Kanten definiert. Existiert kein Weg von s nach t, so definiert man als Länge ∞. Unter Annahme 2.4 existiert ein kürzester Weg, falls der Graph zusammenhängend ist. Lässt man Annahme 2.4 weg, existiert nicht immer ein kürzester Weg, denn wenn z.B. ein Zyklus im Graphen mit negativer Länge existiert, so kann man durch wiederholtes Durchlaufen dieses Zyklus eine Folge von Wegen mit unbeschränkt monoton fallender Länge erhalten.

Um den kürzesten Weg (d.h. den Weg mit der kürzesten Länge) zu finden, kann man bei Graphen, die Annahme 2.4 erfüllen den Algorithmus von Dijkstra[1] verwenden, der schon 1959

[1]Edsger Wybe Dijkstra, 1930-2002, niederländischer Mathematiker und Informatiker

in [4] publiziert wurde und bei effizienter Implementierung noch heute als der effizienteste Algorithmus für dieses Problem bekannt ist. Ein Vergleich verschiedener Algorithmen findet sich in [14] im Abschnitt 7.5b.

Algorithmus 2.5 (Algorithmus von Dijkstra)**.**

Eingabe: *Bewerteter Graph $G = (V, E)$ mit Bewertungsfunktion w, Startpunkt $s \in V$*

Schritt 1: *Setze $\lambda(s) = 0,\ \lambda(v) = \infty\ \forall v \in V \backslash \{s\}$, $T = V$*

Schritt 2: *Falls $T = \emptyset$, so ist der Algorithmus beendet*

Schritt 3: *Wähle einen Knoten $u \in T$ so, dass $\lambda(u)$ minimal ist unter allen $u \in T$*

Schritt 4: *Für jede Kante $e = (u, v)$ mit $v \in T$ setze $\lambda(v) = \min\{\lambda(v), \lambda(u) + w(e)\}$*

Schritt 5: *Setze $T = T \backslash \{u\}$ und gehe zu Schritt 2*

Ausgabe: *$\lambda(v)$ gibt die Länge des kürzesten Weges von s nach v an.*

Der Algorithmus berechnet dabei den kürzesten Weg vom Knoten s zu allen anderen Knoten des Graphen. Falls $\lambda(v) = \infty$ nach Ausführung des Algorithmus ist, so existiert kein Weg vom Startknoten zum Knoten v. Der Aufwand des Algorithmus ist je nach Implementierung $O(|V| \log(|V|) + |E|)$ oder $O(|V|^2)$. Weiteres siehe dazu in [13], Abschnitt 8.5.1. Im Schritt 3 muss ein Knoten u aus T gefunden werden, so dass $\lambda(u)$ minimal ist. Bei der Implementierung ist darauf zu achten, dass das Suchen des Minimums effizient geschieht. Dazu bietet sich eine Heap-Struktur an.
Die Korrektheit des Algorithmus ergibt sich aus folgender Bemerkung.

Bemerkung 2.6. *Sei $p = (v_0, v_1, \ldots, v_k)$ ein kürzester Weg von v_0 nach v_k, so ist jeder Teilweg $p' = (v_i, \ldots, v_j),\ 0 \leq i < j \leq k$, ein kürzester Weg von v_i nach v_j. Denn gäbe es einen kürzeren Weg p'' von v_i nach v_j, so kann der Teilweg p' durch p'' ersetzt werden, was zu einem Widerspruch zur Minimalität von p führt. Daraus folgt, dass die Länge eines kürzesten Weges von s nach v und die Länge einer Kante von v nach v' größer oder gleich der Länge eines kürzesten Weges von s nach v' ist. Des Weiteren folgt, dass für einen kürzesten Weg p' von s nach v', der nicht aus einer Kante besteht, ein Knoten v, eine Kante (v, v') und ein kürzester Weg p von s nach v existiert, so dass die Länge von p' gleich der Länge von p und der Länge der Kante (v, v') ist.*

Beispiel 2.7. *Betrachte folgenden Graphen:*

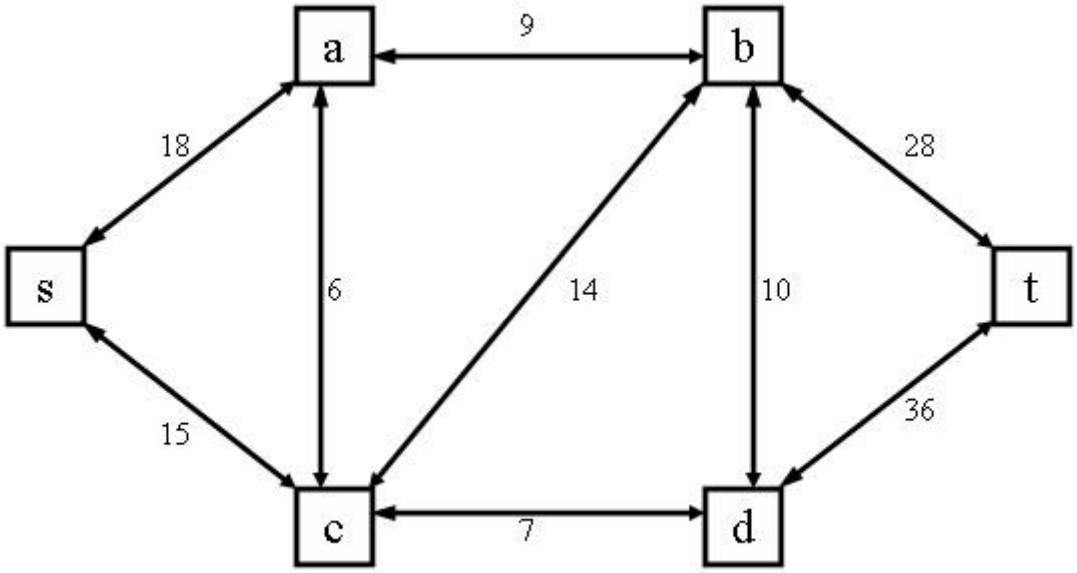

Abbildung 2.1: Graph aus Beispiel 2.6

Schritt 1: Man erhält folgende Situation:

Knoten v	s	a	b	c	d	t
$\lambda(v)$	0	∞	∞	∞	∞	∞
T	{s,	a,	b,	c,	d,	t}

Schritt 3: Wähle $u = s$

Schritt 4: Betrachte die Kanten (s, a) und (s, b). Man erhält $\lambda(a) = 0 + 18 = 0$ und $\lambda(c) = 0 + 15$

Schritt 5: Setze $T = T \backslash \{s\}$

Man erhält

Knoten v	s	a	b	c	d	t
$\lambda(v)$	0	18	∞	15	∞	∞
T	{,	a,	b,	c,	d,	t}

Schritt 3: Wähle $u = c$

Schritt 4: Betrachte die Kanten (c, a), (c, b) und (c, d). Die Kante (c, s) wird nicht betrachtet, da $s \notin T$. Es gilt $\lambda(a) = 18 \leq 21 = 15 + 6 = \lambda(c) + w(c, a)$. Also behält $\lambda(a)$ den Wert 18. Des Weiteren erhält man $\lambda(b) = 15 + 14 = 29$ und $\lambda(d) = 15 + 7$.

Schritt 5: Setze $T = T \backslash \{c\}$

Man erhält

Knoten v	s	a	b	c	d	t
$\lambda(v)$	0	18	29	15	22	∞
T	{	a,	b,		d,	t}

In den nächsten Schritten erhält man

Knoten v	s	a	b	c	d	t
$\lambda(v)$	0	18	27	15	22	∞
T	{		b,		d,	t}

Knoten v	s	a	b	c	d	t
$\lambda(v)$	0	18	27	15	22	58
T	{		b,			t}

Knoten v	s	a	b	c	d	t
$\lambda(v)$	0	18	27	15	22	55
T	{					t}

Schritt 3: Wähle $u = t$

Schritt 4: Es sind keine Kanten zu betrachten, da es außer t keinen anderen Knoten in T gibt.

Schritt 5: Setze $T = \emptyset$

Schritt 2: Ende des Algorithmus

Man erhält also als Ergebnis, dass die Länge des kürzesten Weges von s nach a 18, nach b 27, nach c 15, nach d 22 und nach t 55 ist.

2.2 Weitere Grundbegriffe

In diesem Abschnitt werden einige Grundbegriffe zur Analyse von Kontrollsystemen, die später in der Arbeit verwendet werden, eingeführt.

Definition 2.8. *Sei D eine Teilmenge von $\mathbb{R}^n$ und $f : D \longrightarrow \mathbb{R}^m$ eine Funktion. Dann heißt f* **Lipschitz-stetig** *mit* **Lipschitz-Konstante** *$L > 0$, falls gilt:*

$$\|f(x) - f(y)\| \leq L\|x - y\| \; \forall x, y \in D \tag{2.1}$$

Definition 2.9. *Sei $\alpha : \mathbb{R}_0^+ \longrightarrow \mathbb{R}_0^+$ eine Funktion. Dann heißt α eine* $\mathcal{K}$**-Funktion**, *falls gilt:*

- *α ist stetig.*
- *$\alpha(0) = 0$*
- *α ist streng monoton wachsend.*

Definition 2.10. *Sei $\alpha : \mathbb{R}_0^+ \longrightarrow \mathbb{R}_0^+$ eine Funktion. Dann heißt α eine* $\mathcal{K}_\infty$**-Funktion**, *falls gilt:*

- *α ist eine $\mathcal{K}$-Funktion.*
- *α ist unbeschränkt.*

Definition 2.11. *Sei $\beta : \mathbb{R}_0^+ \times \mathbb{R}_0^+ \longrightarrow \mathbb{R}_0^+$ eine Funktion. Dann heißt β eine* $\mathcal{KL}$**-Funktion**, *falls gilt:*

- *β ist stetig.*
- *$\beta(\cdot, t)$ ist eine $\mathcal{K}$-Funktion $\forall t \geq 0$.*
- *$\beta(r, \cdot)$ ist streng monoton fallend $\forall r \geq 0$.*
- *$\lim_{t\to\infty} \beta(r, t) = 0 \; \forall r \geq 0$.*

Beispiel 2.12.

- *$\alpha(x) = \frac{x}{x+1}$ ist eine $\mathcal{K}$-Funktion.*

- $\alpha(x) = x$ *ist eine* $\mathcal{K}_\infty$*-Funktion.*
- $\beta(r,t) = re^{-t}$ *ist eine* $\mathcal{KL}$*-Funktion.*

Mit Hilfe der $\mathcal{K}$- und$\mathcal{KL}$-Funktionen kann die Stabilität von Kontrollsystemen charakterisiert werden. Näheres siehe dazu im Abschnitt 3.4 von [11].

Kapitel 3

Problemstellung

In dieser Arbeit wird folgendes diskrete Kontrollsystem betrachtet:

$$x_{k+1} = f(x_k, u_k), \quad k = 0, 1, \ldots \tag{3.1}$$

wobei $f : X \times U \longrightarrow \mathbb{R}^d$ eine stetige Funktion und sowohl $X \subset \mathbb{R}^d, \quad 0 \in X$, als auch der Kontrollwertbereich $U \subset \mathbb{R}^m, \quad 0 \in U$, kompakt sind. Des Weiteren wird angenommen, dass das unkontrollierte System in 0 ein Gleichgewicht besitzt, d.h. $f(0, 0) = 0$.
Mit $U^{\mathbb{N}}$ wird die Menge der Kontrollfolgen $\{\mathbf{u} = (u_0, u_1, \ldots), u_i \in U \; \forall i \in \mathbb{N}\}$ bezeichnet.
Mit $(x_k(x, \mathbf{u}))_{k\in\mathbb{N}}$ wird die Trajektorie, die durch

$$x_0(x, \mathbf{u}) = x \;\text{ und }\; x_{k+1}(x, \mathbf{u}) = f(x_k(x, \mathbf{u}), u_k) \;\; k = 0, 1, \ldots$$

definiert ist, bezeichnet, falls $f(x_k(x, \mathbf{u}), u_k) \in X \; \forall k \in \mathbb{N}$.
Der **(schwache) Einzugsbereich** des Gleichgewichtes 0 ist definiert als

$$S = \{x \in X : \exists \mathbf{u} \in U^{\mathbb{N}} \;\text{ mit }\; x_k(x, \mathbf{u}) \to 0, k \to \infty\}$$

Für jedes $x \in X$ sei

$$\mathcal{U}(x) = \{\mathbf{u} \in U^{\mathbb{N}} : \; x_k(x, \mathbf{u}) \to 0, k \to \infty\}$$

die Menge der stabilisierenden Kontrollfolgen von x.
Des Weiteren wird eine Kostenfunktion

$$g : X \times U \longrightarrow \mathbb{R}_0^+$$

betrachtet, wobei g stetig ist, $g(0, 0) = 0$ und $g(x, u) > 0$ für $x \neq 0$ gilt.
Damit wird das Funktional

$$J(x, \mathbf{u}) = \sum_{k=0}^{\infty} g(x_k(x, \mathbf{u}), u_k) \;\in \mathbb{R}_0^+ \cup \{+\infty\} \tag{3.2}$$

definiert.

Mittels dieses Funktionals wird die optimale Wertefunktion dieses Optimierungsproblems als

$$V(x) = \inf_{\mathbf{u} \in \mathcal{U}(x)} J(x, \mathbf{u})$$

definiert.
Mit der Definition $\inf \emptyset = \infty$ ergibt sich $V(x) = \infty$ für $x \notin S$.
Mit S_0 wird die Menge der Punkte x bezeichnet, für die $V(x) < \infty$ gilt.

Satz 3.1 (Bellmann'sches Optimalitätsprinzip)**.** *Die Funktion V erfüllt, falls $V(x) < \infty$ gilt, folgende Gleichung:*

$$V(x) = \inf_{u \in U} \{q(x, u) + V(f(x, u))\} \tag{3.3}$$

Dieses Prinzip ist ein Hauptprinzip der dynamischen Programmierung und wird im Abschnitt 8.1. in [15] behandelt.

Annahme 3.2. *Das Kontrollsystem* (3.1) *ist um die* 0 *lokal asymptotisch kontrollierbar, d.h. es existiert eine $\mathcal{KL}$-Funktion β und eine Umgebung $\mathcal{N}(0) \subset X$ von* 0*, so dass für jeden Punkt $x \in \mathcal{N}(0)$ eine Kontrollfolge $\mathbf{u}$ existiert, die*

$$\|x_k(x, \mathbf{u})\| \leq \beta(\|\mathbf{x}\|, \mathbf{k}) \tag{3.4}$$

erfüllt.

Satz 3.3. *Unter Annahme 3.2 und der Annahme, dass f Lipschitz-stetig und g Lipschitz-stetig und beschränkt ist, ergibt sich die Stetigkeit von V im Einzugsbereich.*

Ein Beweis findet sich im Abschnitt 6.2 von [7].

Bemerkung 3.4.

- *V ist i.A. nicht, auch nicht lokal Lipschitz-stetig.*
- *Unter schwächeren Annahmen erhält man, dass V in einer Umgebung von* 0 *stetig ist.*

Diskrete Kontrollsysteme dieser Art erhält man auch durch eine zeitliche Diskretisierung kontinuierlicher Systeme. Beispielsweise sind solche Systeme durch eine Differentialgleichung folgender Art gegeben:

$$\dot{x}(t) = f(x(t), u(t)) \tag{3.5}$$

wobei $f : X \times U \longrightarrow \mathbb{R}^d$ ein stetiges parameterabhängiges Vektorfeld ist. Die Kosten pro Schritt sind als Integral einer Kostenfunktion, die vom Zustand x und der Kontrolle u abhängt, über die Trajektorie des Systems gegeben.

In dieser Arbeit wird ein effizienter Ansatz zur Berechnung der optimalen Wertefunktion V beschrieben. Ein anderer Ansatz mit Zubovs Methode, der die Prinzipien der dynamischen Programmierung verwendet, findet sich im Abschnitt 6.3 von [7].

Kapitel 4

Graphentheoretischer Ansatz

Das Kontrollsystem (3.1) kann als ein gerichteter Graph

$$G = (X, E), \quad E = \{(x, f(x,u)) : x \in X,\ f(x,u) \in X,\ u \in U\} \tag{4.1}$$

aufgefasst werden. Mit $w(e) = g(x,u) \geq 0$ für $e = (x, f(x,u))$ erhält man einen bewerteten Graphen. Die optimale Wertefunktion $V(x)$ entspricht also der Länge des kürzesten Weges von x nach 0.
Zur Berechnung der optimalen Wertefunktion wird G durch einen endlichen Graphen approximiert.

Definition 4.1. *Eine* **(endliche) Zellenüberdeckung** $\mathcal{P}$ *von* X *ist eine endliche Familie von kompakten Mengen* $P_i,\ i = 1, \ldots, r,$ *mit* $\bigcup_{i=1}^{r} P_i = X,\ P_i = \overline{intP_i}$ *und* $intP_i \cap intP_j = \emptyset$ *für* $i \neq j$.

Beispiel 4.2. *Sei* $X = [0,1] \times [0,1],\ P_1 = [0, 0.5] \times [0, 0.5],\ P_2 = [0.5, 1] \times [0, 0.5],\ P_3 = [0, 0.5] \times [0.5, 1]$ *und* $P_4 = [0.5, 1] \times [0.5, 1]$. *Dann bilden* P_1, P_2, P_3 *und* P_4 *eine Zellenüberdeckung von* X.

Mit einer gegebenen endlichen Zellenüberdeckung kann nun ein endlicher Graph $G_{\mathcal{P}}$ als Approximation von G durch

$$G_{\mathcal{P}} = (\mathcal{P}, E_{\mathcal{P}}), \quad E_{\mathcal{P}} = \{(P_i, P_j) \in \mathcal{P} \times \mathcal{P} : f(P_i, U) \cap P_j \neq \emptyset\}, \tag{4.2}$$

definiert werden, wobei die Kante $e = (P_i, P_j)$ wie folgt bewertet wird:

$$w(e) = \min_{x \in P_i, u \in U} \{g(x,u) : f(x,u) \in P_j\}. \tag{4.3}$$

Da g stetig und $P_i,\ P_j$ und U kompakt sind, wird das Minimum in (4.3) angenommen.

Definition 4.3. *Sei* $p = (e_1, \ldots, e_m)$ *mit* $e_k \in E_{\mathcal{P}}\ \forall k \in \{k = 1, \ldots, m\}, e_k = (p_{k,1}, p_{k,2}), p_{k,2} = p_{k+1,1}\ \forall k \in \{1, \ldots, m-1\}$ *ein Pfad in* $G_{\mathcal{P}}$*, dann wird die* **Länge** *von* p *als*

$$w(p) = \sum_{k=1}^{m} w(e_k) \tag{4.4}$$

definiert.

Definition 4.4. *Mit* $P(x)$ *wird eine Menge* $P \in \mathcal{P}$ *mit* $x \in P$ *bezeichnet. Ein Pfad* $p(x) = (e_1, \ldots, e_m)$ *heißt* **Verbindung von** x **und** 0*, falls gilt:* $e_1 = (P(x), P_1)$ *und* $e_m = (P_{m-1}, P(0))$ *mit* $P_1, \ldots, P_{m-1} \in \mathcal{P}$*.*

Nun wird $V(x)$ durch

$$V_{\mathcal{P}}(x) = \min\{w(p(x)) : p(x) \text{ verbindet } x \text{ und } 0\} \tag{4.5}$$

approximiert.

Die folgenden drei Sätze behandeln Aussagen über die Güte der Approximation, und es wird dargelegt, in welchem Sinne eine Folge von Approximationen gegen die Wertefunktion konvergiert.

Satz 4.5. *Für jede Zellenüberdeckung* $\mathcal{P}$ *von* X *gilt,* $V_{\mathcal{P}}(x) \leq V(x)\ \forall x \in X$*.*

Beweis: Für $x \in X$ mit $V(x) = \infty$ ist die Aussage klar. Sei also $x \in X$ mit $V(x) < \infty$. Es ist ausreichend zu zeigen, dass für jedes $\varepsilon > 0$ und jedes $\mathbf{u} \in U^{\mathbb{N}}$ mit $J(x, \mathbf{u}) < V(x) + \varepsilon$ ein Pfad $p(x)$ in $G_{\mathcal{P}}$ existiert, der x und 0 verbindet, mit $w(p(x)) \leq J(x, \mathbf{u})$. Da die Trajektorie $(x_k(x, \mathbf{u}))_{k \in \mathbb{N}}$ gegen 0 konvergiert, existiert $m \in \mathbb{N}$ so, dass $\forall m' > m\ \ x_{m'}(x, \mathbf{u}) \in P(0)$ gilt, definiert man den Pfad $p(x)$ als

$$p(x) = (e_1, \ldots, e_m), \quad e_k = (P(x_{k-1}), P(x_k)), \quad k = 1, \ldots, m$$

Es gilt

$$\begin{aligned} w(p(x)) &= \sum_{k=1}^{m} w(e_k) = \sum_{k=1}^{m} \min_{x \in P(x_{k-1}), u \in U, f(x,u) \in P(x_k)} \{g(x, u)\} \\ &\leq \sum_{k=1}^{m} g(x_{k-1}, u_{k-1}) \leq \sum_{k=1}^{\infty} g(x_{k-1}, u_{k-1}) = J(x, \mathbf{u}). \end{aligned}$$

□

Definition 4.6. *Sei* $\mathcal{P}$ *eine Zellenüberdeckung von* X*. Dann wird*

$$diam(\mathcal{P}) := \max_{i: P_i \in \mathcal{P}} diam(P_i) \tag{4.6}$$

definiert, wobei diam den Durchmesser einer Zelle bezeichnet.

Um Konvergenz von $V_{\mathcal{P}}$ nach V zu betrachten, wird eine Folge von Zellenüberdeckungen $\mathcal{P}^{(l)}$ mit $\text{diam}(\mathcal{P}^{(l)}) \longrightarrow 0$ für $l \longrightarrow \infty$, so dass für jedes $P_i^{(l+1)} \in \mathcal{P}^{(l+1)}$ ein $P_j^{(l)} \in \mathcal{P}^{(l)}$ mit $P_i^{(l+1)} \subset P_j^{(l)}$ existiert, betrachtet. Damit gilt

Satz 4.7. *Für jedes $x \in S$ ist die Folge $(V_{\mathcal{P}^{(l)}}(x))_{l\in\mathbb{N}}$ monoton wachsend.*

Beweis: Angenommen es existiert ein $x \in S$ mit $V_{\mathcal{P}^{(l+1)}}(x) < V_{\mathcal{P}^{(l)}}(x)$. Sei $p^{(l)}$ und $p^{(l+1)}$ jeweils ein kürzester Pfad von x nach 0 in $G_{\mathcal{P}^{(l)}}$ bzw. $G_{\mathcal{P}^{(l+1)}}$. Daraus folgt $w(p^{(l+1)}) < w(p^{(l)})$. Nun wird ausgehend von $p^{(l+1)}$ ein Pfad $\tilde{p}^{(l)}$ in $G_{\mathcal{P}^{(l)}}$, der x und 0 verbindet, mit $w(\tilde{p}^{(l)}) < w(p^{(l)})$ konstruiert. Dies steht im Widerspruch zur Minimalität von $p^{(l)}$.
Sei $p^{(l+1)} = (e_1^{(l+1)}, \ldots, e_{m(l+1)}^{(l+1)})$ mit $e_k^{(l+1)} = (P_{k-1}^{(l+1)}, P_k^{(l+1)}) \in E_{\mathcal{P}^{(l+1)}}$. Daraus folgt $f(P_{k-1}^{(l+1)}, U) \cap P_k^{(l+1)} \neq \emptyset \ \ \forall k = 1, \ldots, m(l+1)$. Nach Voraussetzung an die Zellenüberdeckungen existieren $\tilde{P}_k^{(l)} \in \mathcal{P}^{(l)}$ mit $P_k^{(l+1)} \subset \tilde{P}_k^{(l)} \ \ \forall k = 0, \ldots, m(l+1)$. Daraus folgt $f(\tilde{P}_{k-1}^{(l)}, U) \cap \tilde{P}_k^{(l)} \neq \emptyset$, also $\tilde{e}_k^{(l)} = (\tilde{P}_{k-1}^{(l)}, \tilde{P}_k^{(l)}) \in E_{\mathcal{P}^{(l)}}$. Damit folgt, $\tilde{p}^{(l)}(x) = (\tilde{e}_1^{(l)}, \ldots, \tilde{e}_{m(l+1)}^{(l)})$ ist ein Pfad in $G_{\mathcal{P}^{(l)}}$, der x und 0 verbindet. Des Weiteren gilt für $k = 1, \ldots, m(l+1)$

$$\begin{aligned} w(\tilde{e}_k^{(l)}) &= \min_{x\in\tilde{P}_{k-1}^{(l)}, u\in U, f(x,u)\in\tilde{P}_k^{(l)}} \{g(x,u)\} \\ &\leq \min_{x\in P_{k-1}^{(l+1)}, u\in U, f(x,u)\in P_k^{(l+1)}} \{g(x,u)\} = w(e_k^{(l+1)}) \end{aligned}$$

Damit ergibt sich $w(\tilde{p}^{(l)}(x)) \leq w(p^{(l+1)}(x)) < w(p^{(l)}(x))$.

□

Damit kann nun der Hauptsatz über die Konvergenz der approximierten Wertefunktion bewiesen werden.

Satz 4.8. *Für jedes $x \in S$ konvergiert die Folge $(V_{\mathcal{P}^{(l)}}(x))_{l\in\mathbb{N}}$ gegen $V(x)$.*

Beweis: Aus Satz 4.5 und 4.7 folgt, dass $V_{\mathcal{P}^{(l)}}(x)$ für alle $x \in S$ konvergiert; mit $\tilde{V}(x)$ wird der Grenzwert bezeichnet.
Angenommen es existiert $x \in S$ mit $\tilde{V}(x) < V(x)$. Nun wird eine Trajektorie $(x_k(x, \mathbf{u}))_{k\in\mathbb{N}}$ konstruiert, die nach 0 konvergiert, mit $J(x, \mathbf{u}) \leq \tilde{V}(x)$. Dies ist ein Widerspruch zur Definition von $V(x)$.
Wegen $g(0, 0) = 0$ kann jeder endliche Pfad, der x und 0 verbindet, durch Hinzufügen der Kante $o^{(l)} = (P^{(l)}(0), P^{(l)}(0))$ zu einem unendlichen Pfad mit derselben Länge ergänzt werden. Sei also $p^{(l)}(x) = (e_k^{(l)})_{k\in\mathbb{N}} := (e_1^{(l)}, \ldots, e_{m(l)}^{(l)}, o^{(l)}, o^{(l)}, \ldots)$, $e_k^{(l)} = (P_{k-1}^{(l)}, P_k^{(l)})$, ein unendlicher minimierender Pfad in $G_{\mathcal{P}^{(l)}}$, der x und 0 verbindet ($l = 0, 1, 2, \ldots$). Es gilt $x_0^{(l)} = x_0 = x \in P_0^{(l)}$ für alle $l \in \mathbb{N}$.
Nun werden der nächsten Punkt x_1 der Trajektorie und der entsprechenden Kontrollwert u_0

konstruiert.
Wähle für jedes $l \in \mathbb{N}$ einen Punkt $\tilde{x}_0^{(l)} \in P_0^{(l)}$ und einen Kontrollwert $u_0^{(l)}$, so dass gilt:

$$g(\tilde{x}_0^{(l)}, u_0^{(l)}) = w(e_1^{(l)})$$

Wegen $\text{diam}(\mathcal{P}^{(l)}) \longrightarrow 0$ für $l \longrightarrow \infty$ gilt $\lim_{l\to\infty} \tilde{x}_0^{(l)} = x_0$. Setze $x_1^{(l)} = f(\tilde{x}_0^{(l)}, u_0^{(l)})$. Da X kompakt ist, existiert eine konvergente Teilfolge $(x_1^{(l)})_{l\in\hat{L}_0}$, $\hat{L}_0 \subset \mathbb{N}$, deren Grenzwert mit x_1 bezeichnet wird. Da U kompakt ist, existiert eine konvergente Teilfolge $(u_0^{(l)})_{l\in L_0}$, $L_0 \subset \hat{L}_0$, deren Grenzwert mit u_0 bezeichnet wird. Wegen der Stetigkeit von f ist $x_1 = f(x_0, u_0)$.
Nun wird diese Konstruktion wiederholt und eine Folge von minimierenden Punkten $(\tilde{x}_k^{(l)})_{l\in L_{k-1}}$ und Kontrollwerten $(u_k^{(l)})_{l\in L_{k-1}}$ gewählt. Man erhält dann den Punkt x_{k+1}, den Kontrollwert u_k mit $x_{k+1} = f(x_k, u_k)$ und eine Menge $L_k \subset L_{k-1}$.
Wähle nun $l_0 \in L_0$ beliebig und setze induktiv l_k als die keinste Zahl aus L_k mit $l_k > l_{k-1}$. Setze nun $L = \{l_0, l_1, \ldots\}$, dann gilt $L \subset L_k \ \forall k \in \mathbb{N}$.
Weiterhin gilt:

$$\lim_{i\to\infty} \tilde{x}_k^{(l_i)} = x_k \ \text{ und } \ \lim_{i\to\infty} u_k^{(l_i)} = u_k.$$

Wegen der Stetigkeit von g gilt:

$$g(x_k, u_k) = \lim_{l\in L} g(\tilde{x}_k^{(l)}, u_k^{(l)}) = \lim_{l\in L} w(e_{k+1}^{(l)})$$

Daraus folgt mit $\mathbf{u} = (u_0, u_1, \ldots)$

$$\begin{aligned} J(x, \mathbf{u}) &= \sum_{k=0}^{\infty} g(x_k, u_k) = \lim_{K\to\infty} \sum_{k=0}^{K} g(x_k, u_k) = \lim_{K\to\infty} \sum_{k=0}^{K} \lim_{l\in L} g(\tilde{x}_k^{(l)}, u_k^{(l)}) \\ &= \lim_{K\to\infty} \sum_{k=0}^{K} \lim_{l\in L} w(e_{k+1}^{(l)}) = \lim_{K\to\infty} \lim_{l\in L} \sum_{k=0}^{K} w(e_{k+1}^{(l)}) \end{aligned}$$

Da $w(e_{k+1}^{(l)}) = 0$ für $k \geq m(l)$ kann man die Summe bei $m(l) - 1$ stoppen und erhält, da $m(l) \geq K$ möglich ist,

$$J(x, \mathbf{u}) \leq \lim_{l\in L} \sum_{k=0}^{m(l)-1} w(e_{k+1}^{(l)}) = \lim_{l\in L} V_{\mathcal{P}^{(l)}}(x) = \tilde{V}(x)$$

Da $J(x, \mathbf{u}) < \infty$, ist $\mathbf{u} \in \mathcal{U}(x)$, denn falls $x_k(x, \mathbf{u})$ nicht gegen 0 konvergiert, existiert eine offene Umgebung N von 0 und eine Teilfolge $(x_k(x, \mathbf{u}))_{k\in M}$, $M \subset \mathbb{N}$, so dass $x_k(x, \mathbf{u}) \notin N \ \forall k \in M$. Da $g(x, u) > 0$ für $x \neq 0$ und g stetig, existiert $c > 0$ mit $g(x, u) > c \ \forall x \in X \setminus N, \ \forall u \in U$. Daher gilt $g(x_k, u_k) > c$ für unendlich viele (x_k, u_k) und somit $J(x, \mathbf{u}) = \infty$.

□

Mit diesen 3 Sätzen erhält man folgendes Korollar.

Korollar 4.9. *Sei $D \subset S$ eine Menge, auf der V stetig ist, und $K \subset D$ kompakt. Dann konvergiert $V_{\mathcal{P}^{(l)}}$ auf K gleichmäßig gegen V.*

Beweis: Sei $\varepsilon > 0$ beliebig gegeben. Wegen der punktweisen Konvergenz aus Satz 4.8 existiert für jedes $x \in K$ ein $n(x) \in \mathbb{N}$, so dass $|V(x) - V_{\mathcal{P}^{(n(x))}}(x)| \leq \frac{\varepsilon}{2}$. Wegen Satz 4.5 kann der Betrag weggelassen werden.

1. Fall Liegt x im Inneren einer Menge aus der Zellenüberdeckung, so ist $V_{\mathcal{P}^{(n(x))}}$ in einer Umgebung von x konstant und wegen der Stetigkeit von V existiert eine offene Umgebung $U(x)$ von x, so dass $\forall y \in U(x)$

$$V(y) - V_{\mathcal{P}^{(n(x))}}(y) \leq \varepsilon \tag{4.7}$$

gilt.

2. Fall Ist x ein Randpunkt einer Menge aus der Zellenüberdeckung, so enthält jede offene Umgebung von x Punkte aus anderen Mengen der Zellenüberdeckung. Da jedoch $V_{\mathcal{P}^{(n(x))}}$ auf den anderen Mengen größer oder gleich $V_{\mathcal{P}^{(n(x))}}(x)$ ist, existiert auch in diesem Fall wegen der Stetigkeit von V eine offene Umgebung $U(x)$ von x, so dass $\forall y \in U(x)$ die Ungleichung (4.7) gilt.

Wegen der Kompaktheit von K existieren $x_1, \ldots, x_m \in K$, so dass $K \subset \bigcup_{i=1}^{m} U(x_i)$ ist. Setze nun $n_0 := \max_{i \in \{1,..m\}} n_0(x_i)$. Damit gilt wegen Satz 4.7

$$V(x) - V_{\mathcal{P}^{(n)}}(x) \leq \varepsilon \; \forall x \in K \; \forall n \geq n_0 \tag{4.8}$$

So ist die gleichmäßige Konvergenz bewiesen.

□

Kapitel 5

Feedback-Regelung und Fehlerschätzer

In diesem Kapitel soll eine Feedback-Regelung hergeleitet werden und die Stabilität der dadurch erzeugten Trajektorien untersucht werden. Dazu wird ein Fehlerschätzer konstruiert.

Aus der dynamischen Programmierung ist bekannt, dass ein optimaler Kontrollwert durch einen Minimierer der rechten Seite von der Gleichung (3.3) (Bellmann'sches Optimalitätsprinzip) gegeben ist. Nun liegt es nahe, eine Feedback-Regelung durch Ersetzen von V durch $V_{\mathcal{P}}$ zu erhalten. Deshalb wird

$$u_{\mathcal{P}}(x) = \arg\min_{u \in U} \{g(x,u) + V_{\mathcal{P}}(f(x,u))\} \tag{5.1}$$

definiert. Das Minimum existiert, da U kompakt ist und $V_{\mathcal{P}}$ nur endlich viele verschiedene Werte annimmt. Der folgende Satz zeigt, in welchem Sinne diese Feedback-Regelung optimal ist.

Satz 5.1. *Gegeben sei eine Folge von verfeinerten Zellenüberdeckungen $(\mathcal{P}^{(l)})_{l \in \mathbb{N}}$ und sei $D \subset S$ eine offene Menge mit folgenden Eigenschaften:*

- $0 \in intD$
- *Für jedes $\varepsilon > 0$ existiert ein $l_0(\varepsilon) \in \mathbb{N}$, so dass*

$$V(x) - V_{\mathcal{P}^{(l)}}(x) \leq \varepsilon \tag{5.2}$$

  für alle $x \in D$ und alle $l \geq l_0(\varepsilon)$ gilt.

Sei $c > 0$ die größte Zahl, so dass die Inklusion $D_c(l) := V^{-1}_{\mathcal{P}^{(l)}}([0,c]) \subset D$ für alle $l \in \mathbb{N}$ gilt. (Bei geeigneter Wahl von $\mathcal{P}^{(1)}$ ergibt sich $c > 0$.)

Dann existiert ein $\varepsilon_0 > 0$ und eine Funktion $\delta : \mathbb{R} \to \mathbb{R}$ mit $\lim_{\alpha \to 0} \delta(\alpha) = 0$, so dass für alle $\varepsilon \in (0, \varepsilon_0]$, für alle $l \geq l_0(\frac{\varepsilon}{2})$, für alle $\eta \in (0,1)$ und alle $x_0 \in D_c(l)$ die Trajektorie, die durch $x_{i+1} = f(x_i, u_{\mathcal{P}^{(l)}}(x_i))$ erzeugt wird, die Ungleichung

$$V(x_i) \leq \max\{V(x_0) - (1-\eta)\sum_{j=0}^{i-1} g(x_j, u_{\mathcal{P}^{(l)}}(x_j)), \delta\left(\frac{\varepsilon}{\eta}\right) + \varepsilon\} \tag{5.3}$$

erfüllt.

Beweis: Für $x \in D$ wird $g_0(x) := \min_{u \in U}\{g(x,u)\}$ definiert. Da V und g_0 in einer Umgebung von 0 stetig sind (vgl. Bemerkung 3.4) und $V(0) = 0$ und $g_0(0) = 0$ gilt, existiert ein $\varepsilon_0 > 0$, so dass aus $g_0(x) < \varepsilon_0$ die Aussage $V(x) \leq c - \varepsilon_0/2$ folgt.
Seien nun $\varepsilon \in (0, \varepsilon_0]$ und $l \geq l_0(\varepsilon/2)$ beliebig.

Behauptung: $x_i \in D_c(l) \Longrightarrow x_{i+1} \in D_c(l)$

Beweis: Sei $x \in D_c(l)$. Dann gilt

$$\begin{aligned} V_{\mathcal{P}^{(l)}}(x) + \varepsilon/2 &\geq V(x) \\ &= \inf_{u \in U}\{g(x,u) + V(f(x,u))\} \\ &\geq \min_{u \in U}\{g(x,u) + V_{\mathcal{P}^{(l)}}(f(x,u))\} \\ &= g(x, u_{\mathcal{P}^{(l)}}(x)) + V_{\mathcal{P}^{(l)}}(f(x, u_{\mathcal{P}^{(l)}}(x))) \\ &= g(x, u_{\mathcal{P}^{(l)}}(x)) + V_{\mathcal{P}^{(l)}}(x_{i+1}) \end{aligned}$$

Daraus folgt

$$\begin{aligned} V_{\mathcal{P}^{(l)}}(x_{i+1}) &\leq V(x_i) - g(x, u_{\mathcal{P}^{(l)}}(x_i)) \\ &\leq V_{\mathcal{P}^{(l)}}(x_i) - g(x, u_{\mathcal{P}^{(l)}}(x_i)) + \varepsilon/2 \end{aligned} \tag{5.4}$$

Fall (a): Falls $g(x, u_{\mathcal{P}^{(l)}}(x_i)) \geq \varepsilon/2$ folgt $V_{\mathcal{P}^{(l)}}(x_{i+1}) \leq V_{\mathcal{P}^{(l)}}(x_i) \leq c$ und somit $x_{i+1} \in D_c(l)$.

Fall (b): Falls $g(x, u_{\mathcal{P}^{(l)}}(x_i)) < \varepsilon/2$, dann ist $g_0(x_i) < \varepsilon/2 \leq \varepsilon_0/2$, daraus folgt $V(x_i) \leq c - \varepsilon_0/2 \leq c - \varepsilon/2$ und somit $V_{\mathcal{P}^{(l)}}(x_{i+1}) \leq V(x_i) + \varepsilon_0/2 \leq c$ und damit auch hier $x_{i+1} \in D_c(l)$

Damit bleibt jede Trajektorie mit $x_0 \in D_c(l)$ für alle Zeiten in $D_c(l)$. Aus (5.4) erhält man folgende Ungleichung:

$$\begin{aligned} V(x_{i+1}) &\leq V_{\mathcal{P}^{(l)}}(x_{i+1}) + \varepsilon/2 \\ &\leq V_{\mathcal{P}^{(l)}}(x_i) - g(x_i, u_{\mathcal{P}^{(l)}}(x_i)) + \varepsilon \\ &\leq V(x_i) - g(x_i, u_{\mathcal{P}^{(l)}}(x_i)) + \varepsilon \end{aligned} \tag{5.5}$$

Nun wird die Funktion $\delta(\alpha)$ konstruiert. Dazu sei $C_\alpha := \{x \in D_c(l) | g_0(x) \leq \alpha\}$ und sei

$$\delta(\alpha) := \sup_{x \in C_\alpha} V(x)$$

definiert. Für $\alpha \to 0$ gilt $\delta(\alpha) \to 0$, da C_α zu 0 zusammenschrumpft und V in der Umgebung von 0 mit $V(0) = 0$ stetig ist.

Behauptung:

$$V(x_i) \leq \delta(\alpha) + \varepsilon \Longrightarrow V(x_{i+1}) \leq \delta(\alpha) + \varepsilon \tag{5.6}$$

gilt für alle $\alpha \geq \varepsilon$.

Beweis: Fall (a) $V(x_i) \leq \delta(\alpha)$: In diesem Fall folgt aus (5.5)

$$\begin{aligned} V(x_{i+1}) &\leq V(x_i) - g(x_i, u_{\mathcal{P}^{(l)}}(x_i)) + \varepsilon \\ &\leq V(x_i) + \varepsilon \leq \delta(\alpha) + \varepsilon \end{aligned}$$

Fall (b) $V(x_i) \in (\delta(\alpha), \delta(\alpha) + \varepsilon]$: In diesem Fall ist $x_i \notin C_\alpha$ und wiederum folgt aus (5.5)

$$\begin{aligned} V(x_{i+1}) &\leq V(x_i) - \underbrace{g(x_i, u_{\mathcal{P}^{(l)}}(x_i))}_{>\alpha \geq \varepsilon} + \varepsilon \\ &\leq V(x_i) \leq \delta(\alpha) + \varepsilon \end{aligned}$$

Schließlich wird die Behauptung des Satzes gezeigt. Sei $\eta \in (0,1)$ und $i \in \mathbb{N}_0$ beliebig. Falls $V(x_i) \leq \delta(\varepsilon/\eta) + \varepsilon$ ist nichts zu zeigen.
Falls $V(x_i) > \delta(\varepsilon/\eta) + \varepsilon$ folgt aus (5.6), dass $V(x_j) > \delta(\varepsilon/\eta) + \varepsilon$ für $i = 0, 1, \ldots, i$, also $x_j \notin C_{\varepsilon/\eta}$ und somit $g(x_j, u_{\mathcal{P}^{(l)}}(x_j)) \geq \varepsilon/\eta$. Daraus folgt $\varepsilon \leq \eta g(x_j, u_{\mathcal{P}^{(l)}}(x_j))$. Damit ergibt sich für $j = 0, 1, \ldots, j-1$ aus (5.5)

$$\begin{aligned} V(x_{j+1}) &\leq V(x_j) - g(x_j, u_{\mathcal{P}^{(l)}}(x_j)) + \varepsilon \\ &\leq V(x_j) - (1-\eta) g(x_j, u_{\mathcal{P}^{(l)}}(x_j)) \end{aligned}$$

Durch Aufaddieren für $j = 0, 1, \ldots, j-1$ erhält man

$$V(x_i) \leq V(x_0) - (1-\eta) \sum_{j=0}^{i-1} g(x_j, u_{\mathcal{P}^{(l)}}(x_j))$$

Damit ist der Satz bewiesen.

□

Um die Güte der Approximation von V durch $V_\mathcal{P}$ zu bewerten, ist es hilfreich, einen Fehlerschätzer zur Verfügung zu haben. Im Folgenden wird ein Fehlerschätzer konstruiert, mit dessen Hilfe der wahre Fehler nach unten abgeschätzt werden kann.

Definition 5.2. *Die Fehlerfunktion $e(x)$ wird auf S_0 folgendermaßen definiert:*

$$e(x) = \min_{u \in U}\{g(x,u) + V_{\mathcal{P}}(f(x,u))\} - V_{\mathcal{P}}(x) \tag{5.7}$$

Folgender Satz besagt, dass $e(x)$ den Fehler nach unten abschätzt.

Satz 5.3. *Sei $\mathcal{P}$ eine Zellenüberdeckung von X. Dann gilt*

$$0 \leq e(x) \leq V(x) - V_{\mathcal{P}}(x) \;\; \forall x \in S_0 \tag{5.8}$$

Beweis: $0 \leq e(x)$ folgt aus der Definition von $V_{\mathcal{P}}$
Andererseits gilt

$$\begin{aligned} V(x) - V_{\mathcal{P}}(x) &= \min_{u \in U}\{g(x,u) + V(f(x,u))\} - V_{\mathcal{P}}(x) \\ &\geq \min_{u \in U}\{g(x,u) + V_{\mathcal{P}}(f(x,u))\} - V_{\mathcal{P}}(x) \\ &= e(x) \end{aligned}$$

□

$e(x)$ liefert eine untere Schranke für den tatsächlichen Fehler. Eine obere Schranke im folgenden Sinne erhält man beim Verfolgen einer Trajektorie.

Satz 5.4. *Es gilt:*

$$V(x_0) - V_{\mathcal{P}}(x_0) \leq \sum_{i=0}^{n} e(x_i) + V(x_{n+1}) - V_{\mathcal{P}}(x_{n+1}) \tag{5.9}$$

Beweis:

$$\begin{aligned} \sum_{i=0}^{n} e(x_i) &= \sum_{i=0}^{n} (\min_{u \in U}\{g(x_i,u) + V_{\mathcal{P}}(f(x_i,u))\} - V_{\mathcal{P}}(x_i)) \\ &= \sum_{i=0}^{n} (g(x_i, u_{\mathcal{P}}(x_i)) + V_{\mathcal{P}}(f(x_i, u_{\mathcal{P}}(x_i))) - V_{\mathcal{P}}(x_i)) \\ &= \sum_{i=0}^{n} (g(x_i, u_{\mathcal{P}}(x_i)) + V_{\mathcal{P}}(x_{i+1}) - V_{\mathcal{P}}(x_i)) \\ &= V_{\mathcal{P}}(x_{n+1}) - V_{\mathcal{P}}(x_0) + \underbrace{\sum_{i=0}^{n} g(x_i, u_{\mathcal{P}}(x_i))}_{\geq V(x_0) - V(x_{n+1})} \\ &\geq V_{\mathcal{P}}(x_{n+1}) - V_{\mathcal{P}}(x_0) + V(x_0) - V(x_{n+1}) \end{aligned}$$

Daraus folgt die Behauptung.

□

Bemerkung 5.5. *Dieses Ergebnis lässt sich leider numerisch nicht ausnützen, denn* $V(x_{n+1})$ *ist unbekannt. Da die Trajektorie nur in eine Umgebung der* 0 *hineinläuft, der* 0 *aber nicht beliebig nahe kommt (vgl. Satz 5.1), lässt sich* $V(x_{n+1})$ *nicht einfach abschätzen.*

Bemerkung 5.6. *Betrachtet man statt der Trajektorie eine Pseudotrajektorie, d.h. folgt man den Zellen, die zu einem kürzesten Weg von der Zelle, die x enthält, zu der Zelle, die* 0 *enthält, gehören, und addiert die maximalen Fehlerschätzer in jeder dieser Zellen auf, erhält man dennoch keine obere Schranke für den tatsächlichen Fehler, wie man anhand des Systems 1 aus dem Kapitel 7 sehen kann. Der Grund dafür liegt darin, dass die Pseudotrajektorie unter Umständen in jeder Zelle springt, d.h. es gibt keine Trajektorie, die der Pseudotrajektorie entspricht.*

Obwohl der Fehlerschätzer $e(x)$ eine untere Schranke für den wahren Fehler angibt, ergibt sich aus dem Fehlerschätzer eine obere Schranke für den Fehler bei Betrachtung der angegebenen Feedbackkontrolle, wie der nächste Satz zeigt.
Dazu wird wieder die Funktion δ aus dem Beweis von Satz 5.1 verwendet.

Satz 5.7. *Gegeben sei eine Zellenüberdeckung* $\mathcal{P}$ *und sei* $D_c = V_{\mathcal{P}}^{-1}([0,c])$ *für ein* $c > 0$. *Falls der Fehlerschätzer e*

$$e(x) \leq \max\{\eta g_0(x), \varepsilon\} \tag{5.10}$$

für alle $x \in D_c$, *ein* $\varepsilon > 0$ *und ein* $\eta \in (0,1)$ *erfüllt, dann erfüllt die Trajektorie, die durch* $x_{i+1} = f(x_i, u_{\mathcal{P}^{(l)}}(x_i))$ *erzeugt wird,*

$$V_{\mathcal{P}}(x_i) \leq \max\{V_{\mathcal{P}}(x_0) - (1-\eta)\sum_{j=0}^{i-1} g(x_j, u_{\mathcal{P}}(x_j)), \delta\left(\frac{\varepsilon}{\eta}\right) + \varepsilon\} \tag{5.11}$$

für alle $x_0 \in D_c$.

Beweis: Falls $V_{\mathcal{P}}(x) \leq \delta(\varepsilon/\eta) + \varepsilon$ für alle $x \in D_c$ gilt, ist nichts zu zeigen.
Also kann angenommen werden, dass $V_{\mathcal{P}}(x) > \delta(\varepsilon/\eta) + \varepsilon$ für ein $x \in D_c$ gilt, daraus folgt $c > \delta(\varepsilon/\eta) + \varepsilon$. Aus der Definition von δ folgt

$$g_0(x) \leq \alpha \Rightarrow V(x) \leq \delta(\alpha) \Rightarrow V_{\mathcal{P}}(x) \leq \delta(\alpha)$$

für jedes $\alpha \geq 0$.

Behauptung: $x_i \in D_c \Longrightarrow x_{i+1} \in D_c$

Beweis: Sei $x_i \in D_c(l)$. Dann folgt aus der Definition des Fehlerschätzers

$$V_{\mathcal{P}}(x_{i+1}) \leq V_{\mathcal{P}}(x_i) - g(x_i, u_{\mathcal{P}}(x_i)) + e(x_i) \tag{5.12}$$

für alle $x_i \in D_c$.

Fall (a) Falls $g(x_i, u_{\mathcal{P}}(x_i)) \geq e(x_i)$, erhält man $V_{\mathcal{P}}(x_{i+1}) \leq V_{\mathcal{P}}(x_i) \leq c$, und somit $x_{i+1} \in D_c$.

Fall (b) Falls $g(x_i, u_{\mathcal{P}}(x_i)) < e(x_i)$, so ist $\eta g_0(x_i) < e(x_i)$, somit $e(x_i) < \varepsilon$ und $g_0(x_i) < \varepsilon/\eta$. Daraus folgt $V_{\mathcal{P}}(x_i) \leq \delta(\varepsilon/\eta)$ und somit $V_{\mathcal{P}}(x_{i+1}) \leq V_{\mathcal{P}}(x_i) + \varepsilon \leq \delta(\varepsilon/\eta) + \varepsilon \leq c$. Damit ist auch hier $x_{i+1} \in D_c$ gezeigt.

Behauptung:

$$V_{\mathcal{P}}(x_i) \leq \delta(\varepsilon/\eta) + \varepsilon \Longrightarrow V_{\mathcal{P}}(x_{i+1}) \leq \delta(\varepsilon/\eta) + \varepsilon \tag{5.13}$$

gilt.

Beweis: Falls $e(x_i) \leq \eta g_0(x_i)$ folgt aus (5.12) $V_{\mathcal{P}}(x_{i+1} \leq V_{\mathcal{P}}(x_i)$ und somit die Behauptung. Falls $e(x_i) > \eta g_0(x_i)$, muss $e(x_i) \leq \varepsilon$ gelten.

Fall (a) Falls $V_{\mathcal{P}}(x_i) \leq \delta(\varepsilon/\eta)$ folgt aus (5.12)

$$\begin{aligned} V_{\mathcal{P}}(x_i) &\leq V_{\mathcal{P}}(x_i) - g(x_i, u_{\mathcal{P}}(x_i)) + \varepsilon \\ &\leq V_{\mathcal{P}}(x_i) + \varepsilon \leq \delta(\varepsilon/\eta) + \varepsilon \end{aligned}$$

Fall (b) $V_{\mathcal{P}}(x_i) \in (\delta(\varepsilon/\eta), \delta(\varepsilon/\eta) + \varepsilon]$: In diesem Fall ist $x_i \notin C_{\varepsilon/\eta}$ und wiederum folgt aus (5.12)

$$\begin{aligned} V_{\mathcal{P}}(x_{i+1}) &\leq V_{\mathcal{P}}(x_i) - \underbrace{g(x_i, u_{\mathcal{P}}(x_i))}_{>\varepsilon/\eta \geq \varepsilon} + \varepsilon \\ &\leq V_{\mathcal{P}}(x_i) \leq \delta(\varepsilon/\eta) + \varepsilon \end{aligned}$$

Schließlich wird die Behauptung des Satzes gezeigt. Sei $i \in \mathbb{N}_0$ beliebig. Falls $V_{\mathcal{P}}(x_i) \leq \delta(\varepsilon/\eta) + \varepsilon$ ist nichts zu zeigen.
Falls $V_{\mathcal{P}}(x_i) > \delta(\varepsilon/\eta) + \varepsilon$ folgt aus (5.13), dass $V_{\mathcal{P}}(x_j) > \delta(\varepsilon/\eta) + \varepsilon$ für $i = 0, 1, \ldots, i$, also $x_j \notin C_{\varepsilon/\eta}$ und somit $g(x_j, u_{\mathcal{P}}(x_j)) \geq \varepsilon/\eta$. Daraus folgt $\varepsilon(x_j) \leq \eta g(x_j, u_{\mathcal{P}}(x_j))$. Damit ergibt sich für $j = 0, 1, \ldots, j-1$ aus (5.12)

$$\begin{aligned} V_{\mathcal{P}}(x_{j+1}) &\leq V_{\mathcal{P}}(x_j) - g(x_j, u_{\mathcal{P}}(x_j)) + e(x_j) \\ &\leq V_{\mathcal{P}}(x_j) - (1-\eta) g(x_j, u_{\mathcal{P}}(x_j)) \end{aligned}$$

Durch Aufaddieren für $j = 0, 1, \ldots, j-1$ erhält man

$$V_{\mathcal{P}}(x_i) \leq V_{\mathcal{P}}(x_0) - (1-\eta) \sum_{j=0}^{i-1} g(x_j, u_{\mathcal{P}}(x_j))$$

Damit ist der Satz bewiesen.

□

Folgende Bemerkung weist auf die Bedeutung dieses Satzes hin.

Bemerkung 5.8. *Im Satz 5.7 werden zwei Parameter ε und η verwendet. Der Parameter ε bestimmt eine obere Schranke für den kleinsten erreichbaren Wert $V_{\mathcal{P}}(x_i)$, während der Parameter η die Geschwindigkeit des Abstieges von $V_{\mathcal{P}}(x_i)$ festlegt. Wählt man η sehr klein und ε so, dass auch η/ε sehr klein, erhält man eine gute Approximation, dazu ist aber eine sehr feine Zellenüberdeckung möglich. Es empfiehlt sich u.U. η relativ groß zu wählen, da dann die Trajektorien zwar langsamer, aber näher zum Gleichgewicht laufen, und eine gröbere Überdeckung ausreicht, was wiederum deutlich Rechenzeit spart.*

Der Satz erlaubt für Sampled-Data-Systeme eine weitergehende Aussage.

Bemerkung 5.9. *Es kann gezeigt werden, dass dieser Ansatz unter der Annahme des Satzes 5.7 eine praktisch asymptotisch stabile Kontrolle für Sampled-Data-Systemen liefert. Näheres siehe dazu bei Korollar 1 von [8] in Verbindung mit [12]. Dies ist ein wichtiges Ergebnis, denn bei technischen Anwendungen ergeben sich oft Sampled-Data-Systeme, da ein technisches System meistens nicht kontinuierlich beobachtet und geregelt werden kann. Deshalb wird das System zu einem Zeitpunkt beobachtet und anhand der Beobachtung der Kontrollwert bestimmt, der auf dem nächsten Zeitintervall wirkt.*

Kapitel 6

Implementierung

6.1 Allgemeines

Die im Kapitel 4 vorgestellte Approximation an $V(x)$ lässt sich nicht direkt programmieren, da die Kantenmenge und die Bewertung der Kanten nicht errechnet werden kann, weil dazu eine unendliche Anzahl von Punkten betrachtet werden müßte. Deshalb wird eine endliche Zahl von Testpunkten verwendet, um $f(P_i, U)$ zu bestimmen und für $e = (P_i, P_j) wird$

$$w(e) = \min_{(x,u)\in T_i} \{g(x,u) : f(x,u) \in P_j\}$$

gesetzt, wobei $T_i \subset P_i \times U$ eine endliche Menge von Testpunkten ist. Damit ist Satz 4.5 nicht mehr garantiert, dennoch liefert der Algorithmus auch bei wenigen Testpunkten in der Praxis gute Ergebnisse, besonders wenn Randpunkte der Zellen zu den Testpunkten dazugenommen werden.

Implementierung der Zellenverwaltung
In dieser Arbeit werden Rechtecke bzw. Hyperquader in höheren Dimensionen als Zellen verwendet. Zur Verwaltung der Zellen wird der Ansatz aus [3] zu Grunde gelegt. Diese Datenstruktur ist speichersparend und schnell, da die einzelnen Zellen in einem binären Baum verwaltet werden. Die Implementierung im Rahmen dieser Arbeit beruht auf einer Implementierung von Prof. Grüne, die sich in [6] findet und für den graphentheoretischen Ansatz erweitert wurde.

Implementierung des Graphen
In dieser Arbeit wurde die Datenstruktur der einzelnen Zellen so modifiziert, dass sowohl die Kanten, die von einer Zelle ausgehen, mit den Gewichten als verkettete Listen als auch die Kanten, die zu einer Zelle hinführen, (ohne die Gewichte) als verkettete Listen gespeichert werden. Dies erlaubt bei Verfeinerung der Zellenüberdeckung, dass Teile des Graphen wiederverwendet werden können, also nicht neu berechnet werden müssen.

Implementierung des Dijkstra-Algorithmus
In dieser Arbeit wird die Menge T aus dem Dijkstra-Algorithmus als binärer Heap implementiert, um den Aufwand als $O(|V|\log(|V|) + |E|)$ zu erhalten. Diese Implementierung bietet sich an, da die Zahl der Kanten im Graphen etwa proportional zu der Zahl der Knoten ist.

Implementierung des Differentialgleichungslösers
Um zeitlich diskretisierte Systeme, die durch Differentialgleichungen gegeben sind, betrachten zu können, ist die numerische Lösung von Differentialgleichungen nötig. Hierzu wird ein selbst implementiertes adaptives eingebettetes Runge-Kutta-Verfahren der Ordnung (4,5), wie im Abschnitt 2.7 von [5] beschrieben, verwendet.

Adaptive Verfeinerungsstrategie
In anderen Bereichen der numerischen Mathematik (z.B. beim numerischen Lösen gewöhnlicher Differentialgleichungen) wird ein Fehlerschätzer eingesetzt, um eine adaptive Verfeinerungsstrategie zu erhalten. In dieser Arbeit liegt jedoch nur ein Schätzer für eine untere Schranke des wahren Fehlers vor.
Eine Idee, um eine adaptive Verfeinerungsstrategie mit Hilfe des Fehlerschätzers $e(x)$ zu erhalten, ist folgende: Man vergleicht $e(x)$ mit $g(x, u_{\mathcal{P}}(x))$. Aus $e(x) = \min_{u\in U}\{g(x,u) + V_{\mathcal{P}}(f(x,u))\} - V_{\mathcal{P}}(x) = g(x, u_{\mathcal{P}}(x)) + V_{\mathcal{P}}(f(x, u_{\mathcal{P}}(x))) - V_{\mathcal{P}}(x) < g(x, u_{\mathcal{P}}(x))$ folgt nämlich:

$$V_{\mathcal{P}}(f(x, u_{\mathcal{P}}(x))) < V_{\mathcal{P}}(x).$$

Somit ist dann garantiert, dass die Wertefunktion entlang der Trajektorie abnimmt.

Eine Beschreibung der einzelnen Dateien der Implementierung befindet sich im Anhang A.

6.2 Dokumentation der einzelnen Funktionen

Als erstes werden die Funktionen der Impementierung aus [6] beschrieben. Diese Dokumentation ist aus dem Anhang von [6] entnommen.

- `double make_grid(double *xu, double *xo, int dim, int r);`

  Erzeugt eine Zellenüberdeckung im $\mathbb{R}^{dim}$ auf der rechteckigen Menge Ω mit "linker unterer" Ecke xu und "rechter oberer" Ecke xo. In jeder Koordinatenrichtung werden r Zellen erzeugt, also $r \cdot dim$ Zellen insgesamt. Falls r keine Zweierpotenz ist wird automatisch abgerundet. Als Rückgabewert gibt die Funktion den Durchmesser der Zellen (in der 2-Norm) zurück. Jede Zelle wird mit Status=2 vorbelegt.

  Beispiel (2d):

  ```
  double k, xu[2], xo[2];
  xu[0] = -1; xu[1] = -1;
  xo[0] = 1; xo[1] = 1;
  k=make_grid(xu,xo,2,4);
  ```

- `int write_grid(char *filename);`

  Schreibt die aktuelle Zellenüberdeckung in die Datei mit Namen `filename`. Für jede Zelle wird eine Zeile mit linker unterer und rechter oberer Ecke sowie dem Status ausgegeben.

  Beispiel:

  ```
  write_grid("test.grd");
  ```

- `void refine_grid(void);`

  Verfeinert jede Zelle des Gitters mit Status=2. Hierbei wird immer in eine Koordinatenrichtung verfeinert, wobei sich die Richtungen zyklisch abwechseln, in 3d also x - y - z - x - y - z - ...

- `int first_cell(void);`

  Setzt den internen Zeiger auf die erste Zelle der Überdeckung. Gibt -1 zurück, falls noch kein Gitter erzeugt wurde, sonst 0. Die Zelle, auf die der interne Zeiger zeigt, wird im Folgenden als aktuelle Zelle bezeichnet.

- `int next_cell(void);`

  Setzt den internen Zeiger auf die nächste Zelle der Überdeckung. Gibt eine fortlaufenden positiven Nummer (Index der Zelle) zurück und -1, falls die letzte Zelle der Überdeckung bereits erreicht war.

  Beispiel für `first_cell` und `next_cell`: Schleife über alle Zellen

```
int index;
...
index = first_cell();
do
{
...
/* Hier kann die aktuelle Zelle bearbeitet werden */
...
index = next_cell();
}
while (index!=-1);
```

- `void get_corner(int i, double *x);`

  Gibt den i-ten Eckpunkt der aktuellen Zelle (siehe `first_cell`) aus. Der Index i muss zwischen 0 und $2^{dim} - 1$ liegen, die Variable x muss entsprechend der Dimension deklariert sein.

  Beispiel (2d):

```
double x[2];
int i;
...
for (i=0; i<4; i++)
{
get_corner(i, x);
printf("Eckpunkt %d: %f %f\n", i, x[0], x[1]);
}
```

- `void get_testpoint(int i, int tpnum, double *x);`

  Gibt den i-ten Testpunkt (von insgesamt $tpnum$) in der aktuellen Zelle aus. Der Index i muss zwischen 0 und $tpnum-1$ liegen, die Variable x muss entsprechend der Dimension deklariert sein. Die Testpunktanzahl muss $\geq 2^{dim}$ sein und sollte von der Form m^{dim} für ein $m \in \mathbb{N}$ sein, falls dies nicht der Fall ist, wird automatisch intern abgerundet. Die Testpunkte werden gleichmäßig in der Zelle platziert; dabei sind die Eckpunkte immer Testpunkte.

  Beispiel (2d):

```
double x[2];
int i,j;
...
j = 9;
```

```
for(i=0; i<j; i++)
{
get_testpoint(i,j,x);
printf("Testpunkt %d: %f %f\n", i, x[0], x[1]);
}
```

- `int get_status(void);`

  Gibt den Status der aktuellen Zelle zurück bzw. -1, falls der interne Zeiger nicht gesetzt wurde.

- `int set_status(int status);`

  Setzt den Status der aktuellen Zelle auf *status*; dieser Wert muss dabei ≥ 0 sein. Gibt bei Erfolg 0 zurück, sonst -1.

- `int get_x_status(double *x);`

  Gibt den Status der Zelle zurück, in der der Punkt x liegt bzw. -1, falls x nicht in Ω liegt oder keine Überdeckung erzeugt wurde.

- `int set_x_status(double *x, int status);`

  Setzt den Status der Zelle, in der x liegt auf *status*; dieser Wert muss dabei ≥ 0 sein. Gibt bei Erfolg 0 zurück, sonst -1.

- `int set_xngh_status(double *x, int status);`

  Wie `int set_x_status`, wobei zudem alle benachbarten Zellen markiert werden (auch wenn x außerhalb von Ω liegt).

Als nächstes werden die selbst erstellten Funktionen für den graphentheoretischen Ansatz zur Berechnung der optimalen Wertefunktion und für die Feedback-Kontrolle beschrieben.

Ein Kontrollsystem wird in dieser Implementierung durch eine Funktion `void f(double *x, double *y, double u, int dim, double *gewicht)` dargestellt, wobei in y nach der Auswertung der Funktion der Funktionswert in abhängig von x und u steht. Die Kosten werden in der Variable, auf die *gewicht* zeigt, gespeichert.
Beispiel:
Das Kontrollsystem

$$x_{k+1} = x_k + \frac{1}{2}u_k x_k,\ g(x,u) = \frac{1}{2}x$$

wird folgendermaßen dargestellt:

```
void f(double *x, double *y, double u, int dim, double *gewicht)
{
 y[0]=x[0]+0.5*u*x[0];
 *gewicht=0.5*x[0];
}
```

Nun folgen die neuen Funktionen:

- `void erzeuge_graph(int dim, int anzahl_u, double *U, int tp, void f(double *x, double *y, double u, int dim, double *gewicht));`

  Diese Funktion erzeugt den Graphen für das Kontrollsystem f der Dimension dim, wobei $anzahl\_u$ Kontrollwerte, die im Array U gespeichert sind, und tp Testpunkte verwendet werden. Für tp gilt analog die Anforderung wie bei `void get_testpoint(int i, int tpnum, double *x);`

- `void berechne_wertefunktion(int dim);`

  Diese Funktion berechnet mit dem Dijkstra-Algorithmus die Wertefunktion für das dim-dimensionale Kontrollsystem. Die Funktion `void erzeuge_graph(int dim, int anzahl_u, double *U, int tp, void f(double *x, double *y, double u, int dim, double *gewicht));` muss vorher aufgerufen worden sein.

- `void ausgabe_wertefunktion(int dim, char* dname);`

  Diese Funktion gibt die Wertefunktion in die Datei mit Namen $dname$ aus. Dabei wird für jede Zelle eine Zeile mit der linken unteren und der oberen rechten Ecke und dem Wert der Wertefunktion auf der Zelle ausgegeben.

  Beispiel zu den letzten drei Funktionen:

  ```
  int dim = 1;
  int tp = 8;
  double *xu, *xo;
  int z=16;
  int anzahl_u=21;
  double *U;

  ...
  /*
  Speicheranfordern fuer xu, xo, U
  Initialisieren von xu, xo, U
  */
  ```

```
make_grid(xu,xo,dim,z);
erzeuge_graph(dim,anzahl_u,U,tp,f);
berechne_wertefunktion(dim);
ausgabe_wertefunktion(dim,"werte.dat");
```

- `double get_wert(void);`

  Diese Funktion gibt den Wert der Wertefunktion auf der aktuellen Zelle zurück. Falls der interne Zeiger nicht gesetzt ist, wird -1 zurückgegeben.

- `double get_x_wert(double *x);`

  Diese Funktion gibt den Wert der Wertefunktion auf der Zelle, die x enthält, zurück. Falls x nicht in der Zellenüberdeckung enthalten ist, wird -1 zurückgegeben.

- `void update_graph(int dim, int anzahl_u, double *U, int tp,`
  `void f(double *x, double *y, double u, int dim, double *gewicht));`

  In dieser Funktion werden alle Zellen mit Status 2 verfeinert und der Graph wird teilweise neu berechnet, so dass der Graph dann der neuen Zellenüberdeckung entspricht. Dazu werden wieder tp Testpunkte pro Zelle verwendet. Danach wird die Wertefunktion neu berechnet.

- `void ausgabe_trajektorie(int dim, int anzahl_u, double *U, int k,`
  `void f(double *x, double *y, double u, int dim, double *gewicht),`
  `double* x, char* dname);`

  Diese Funktion schreibt die durch die Feedback-Kontrolle erzeugte Trajektorie mit Startwert x über k Schritte in die Datei mit den Namen *dname*. Dabei wird pro Zeile ein Punkt der Trajektorie ausgegeben.

- `double get_x_fehler(double *x, int anzahl_u, double *U, int dim,`
  `void f(double *x, double *y, double u, int dim, double *gewicht));`

  Diese Funktion gibt den Fehlerschätzer $e(x)$ zurück, falls x in Ω liegt, ansonsten -1.

- `double get_x_g_fehler(double *x, int anzahl_u, double *U, int dim,`
  `void f(double *x, double *y, double u, int dim, double *gewicht),`
  `double *g_m);`

  Diese Funktion liefert den Fehlerschätzer $e(x)$ zurück und in der Variable, auf die g_m zeigt, wird das minimale $g(x, u)$ gespeichert. Wiederum gibt diese Funktion -1 zurück, falls x nicht in Ω liegt.

Kapitel 7

Beispiele und numerische Ergebnisse

7.1 Ein einfaches ein-dimensionales Beispiel

In diesem Abschnitt wird folgendes System betrachtet:

$$x_{k+1} = x_k + (1-a)u_k x_k, \quad k = 0, 1, \ldots, \tag{7.1}$$

wobei $x \in X = [0,1], u_k \in U = [-1,1]$ und $a \in (0,1)$ ein fester Parameter ist.
Als Kostenfunktion wird

$$g(x,u) = (1-a)x$$

betrachtet.
Da $g(x,u)$ nicht von u abhängt, ist es am besten, das System möglichst schnell zur 0 zu steuern. Dies erreicht man durch die konstante Kontrollfolge $\mathbf{u} = (-1,-1,-1,\ldots)$.
Mit dieser Kontrollfolge erhält man

$$x_{k+1} = x_k + (1-a)(-1)x_k = x_k - x_k + ax_k = ax_k$$

und induktiv

$$x_k = a^k x_0.$$

Somit gilt

$$V(x) = \sum_{k=0}^{\infty} g(x_k, -1) = \sum_{k=0}^{\infty} (1-a)a^k x = x(1-a)\sum_{k=0}^{\infty} a^k = x(1-a)\frac{1}{1-a} = x$$

Für die Berechnungen werden 8 Testpunkte pro Zelle verwendet, U wird durch 21 äquidistante Punkte approximiert und der Parameter a wird als 0.8 gewählt.

Bei gleichmäßiger Verfeinerung des Gitters erhält man folgende Approximationen $V_{\mathcal{P}}$, zum Vergleich ist jeweils die wahre optimale Wertefunktion $V(x) = x$.

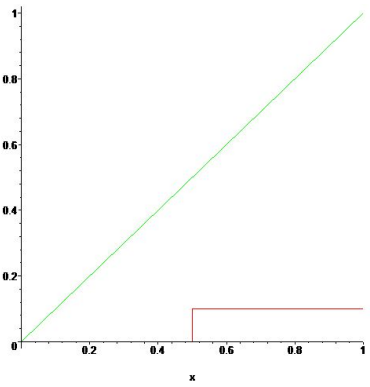

Abbildung 7.1: $V_{\mathcal{P}}$ mit 2 Zellen

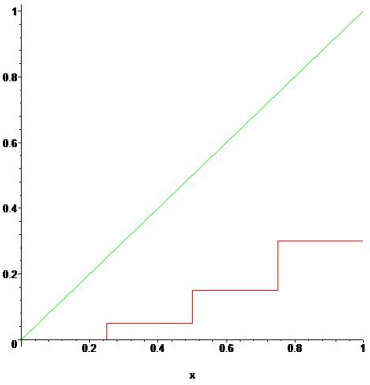

Abbildung 7.2: $V_{\mathcal{P}}$ mit 4 Zellen

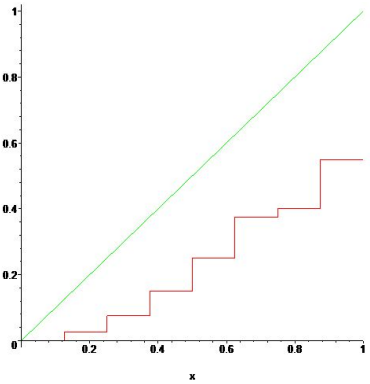

Abbildung 7.3: $V_{\mathcal{P}}$ mit 8 Zellen

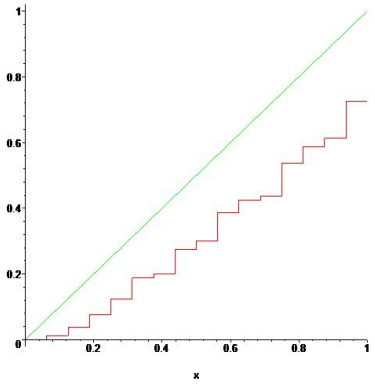

Abbildung 7.4: $V_{\mathcal{P}}$ mit 16 Zellen

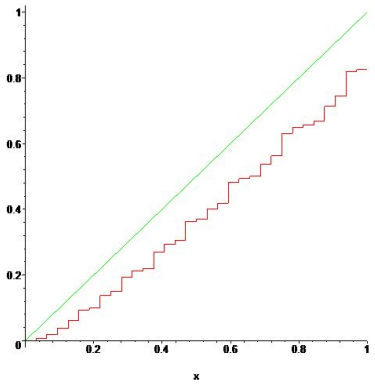

Abbildung 7.5: $V_{\mathcal{P}}$ mit 32 Zellen

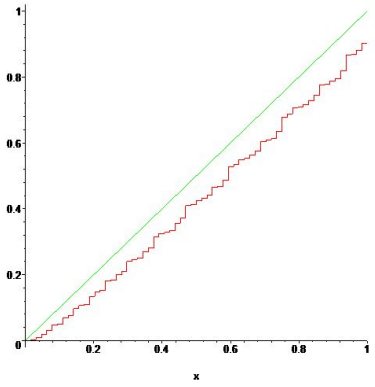

Abbildung 7.6: $V_{\mathcal{P}}$ mit 64 Zellen

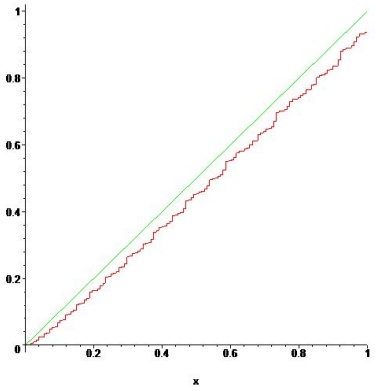

Abbildung 7.7: $V_{\mathcal{P}}$ mit 128 Zellen

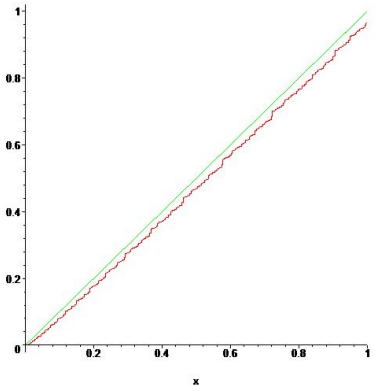

Abbildung 7.8: $V_{\mathcal{P}}$ mit 256 Zellen

Die nächste Abbildung zeigt bei einer Überdeckung mit 64 Zellen den Fehlerschätzer $e(x)$ (rot) im Vergleich zum wahren Fehler (grün).

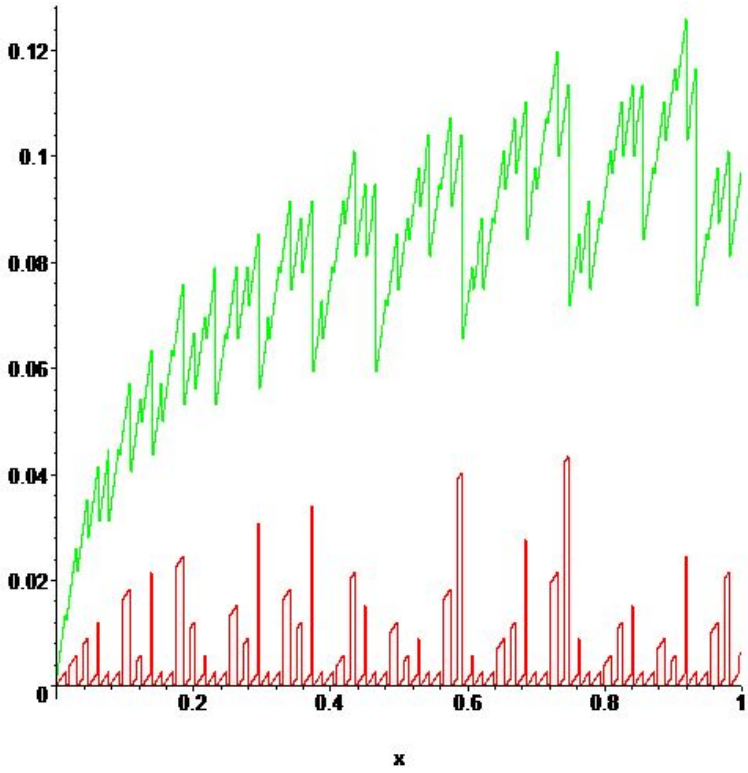

Abbildung 7.9: Vergleich zwischen $e(x)$ und dem wahren Fehler

Die Feedback-Regelung wählt bei diesem Beispiel immer den Kontrollwert -1, obwohl bei diesem Beispiel der Kontrollwert nicht eindeutig ist, da $g(x, u)$ nicht von u abhängt. Somit konvergieren die Trajektorien gegen 0.

7.2 Pendelmodell

Als zweites Beispiel wird ein Pendelmodell angeführt, das auch in [10] und [8] untersucht wurde.
Ein Wagen, auf dem ein umgedrehtes starres Pendel befestigt ist, kann beschleunigt oder abgebremst werden. Dies stellt die Kontrolle des Systemes dar. Das Ziel ist es, durch günstiges Beschleunigen und Abbremsen das Pendel aufzurichten und in der Senkrechten zu halten. Daraus ergibt sich folgendes Kontrollsystem:

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= \frac{\frac{g}{l}\sin(x_1) - \frac{1}{2}m_r x_2^2 \sin(2x_1) - \frac{m_r}{ml}\cos(x_1)u}{\frac{4}{3} - m_r\cos^2(x_1)} \end{aligned} \tag{7.2}$$

Hierbei stellt M die Masse des Wagens, m die Masse des Pendels, $m_r = m/(m+M)$ das Massenverhältnis, l die Entfernung des Schwerpunktes des Pendels vom Wagen und g=Erdanziehungskraft dar. x_1 entspricht dem Winkel des Pendels zur Senkrechten und x_2 der Winkelgeschwindigkeit. Die Position und Geschwindigkeit des Wagens wird hier nicht betrachtet.
Als Kostenfunktion wird

$$q(x,u) = \frac{1}{2}(0.1x_1^2 + 0.05x_2^2 + 0.01u^2) \tag{7.3}$$

verwendet.
Als Parameter werden $M = 8$, $m = 2$, $l = 0.5$ und $g = 9.8$ verwendet.
Durch

$$f(x,u) = \phi^T(x;u) \tag{7.4}$$

erhält man ein zeitdiskretes Kontrollsystem, dabei bezeichnet $\phi^t(x;u)$ die Lösung von (7.2) mit Anfangswert x und konstanter Steuerung u zur Zeit t.
Die Kostenfunktion $g(x,u)$ ist durch

$$g(x,u) = \int_0^T q(\phi^t(x;u),u)dt \tag{7.5}$$

gegeben.
Hier wird $T = 0.1$, $X = [-8,8]\times[-10,10]$ und $U = \{-64,-56,\ldots,-8,0,8,\ldots,56,64\}$verwendet.
Für alle Berechnungen werden 9 Testpunkte pro Zelle verwendet.

Die Farben in den nachfolgenden Bildern stellen den Wert der berechneten Wertefunktion dar. Die Werte steigen von Blau zu Rot an. Falls $V_{\mathcal{P}}(x) > 7$ ist, so wird hier $V_{\mathcal{P}}(x)$ als 7 ausgegeben.

Bei gleichmäßiger Verfeinerung des Gitters erhält man folgende Approximationen $V_{\mathcal{P}}$.

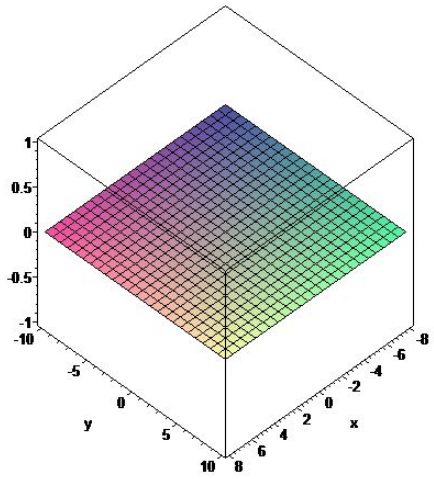

Abbildung 7.10: $V_{\mathcal{P}}$ mit 4 Zellen

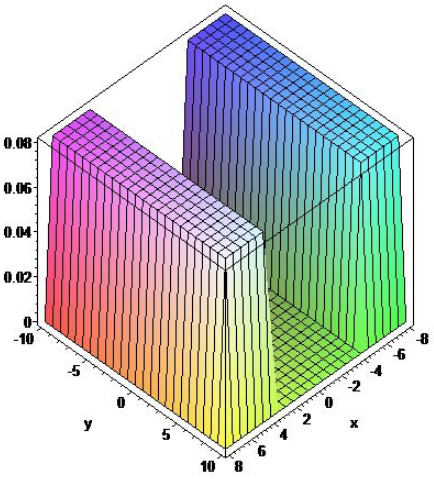

Abbildung 7.11: $V_{\mathcal{P}}$ mit 8 Zellen

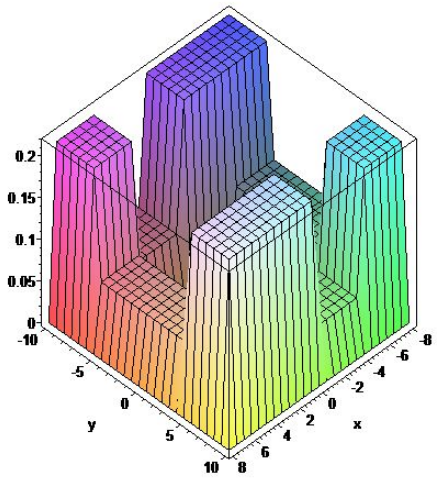

Abbildung 7.12: $V_{\mathcal{P}}$ mit 16 Zellen

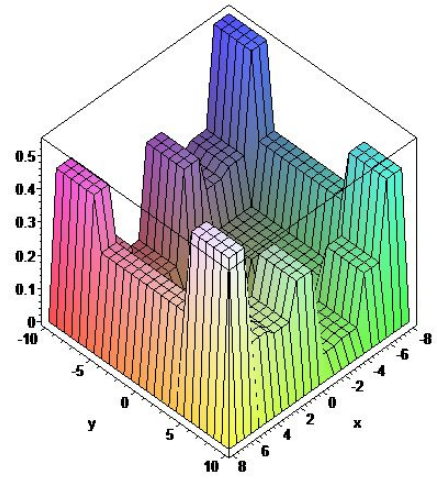

Abbildung 7.13: $V_{\mathcal{P}}$ mit 32 Zellen

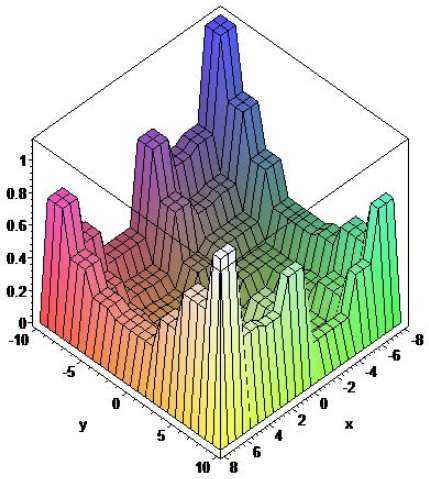

Abbildung 7.14: $V_{\mathcal{P}}$ mit 64 Zellen

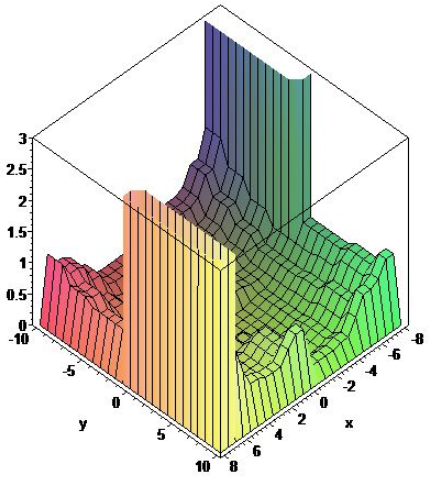

Abbildung 7.15: $V_{\mathcal{P}}$ mit 128 Zellen

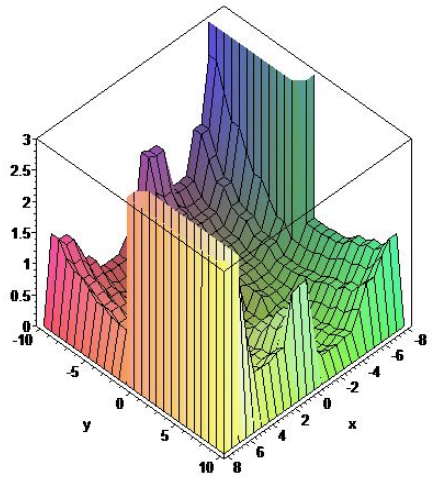

Abbildung 7.16: $V_{\mathcal{P}}$ mit 256 Zellen

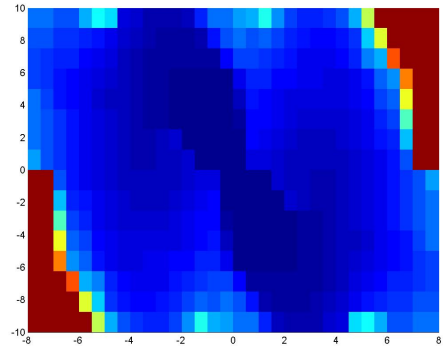

Abbildung 7.17: $V_{\mathcal{P}}$ mit 512 Zellen

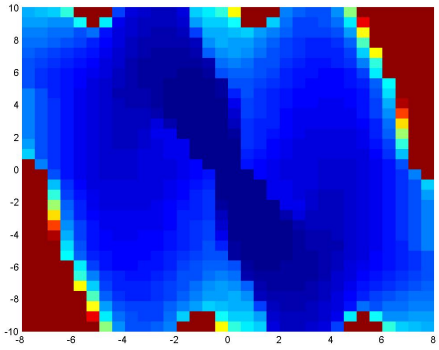

Abbildung 7.18: $V_{\mathcal{P}}$ mit 1024 Zellen

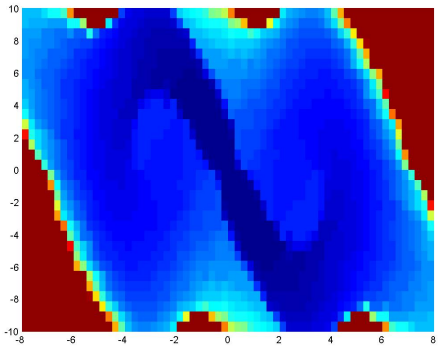

Abbildung 7.19: $V_{\mathcal{P}}$ mit 2048 Zellen

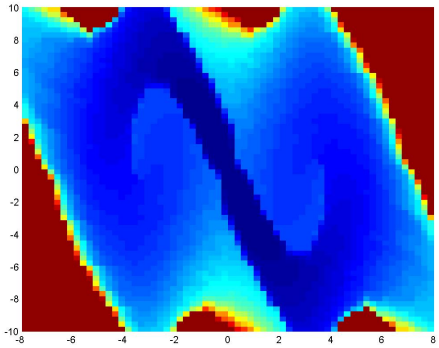

Abbildung 7.20: $V_{\mathcal{P}}$ mit 4096 Zellen

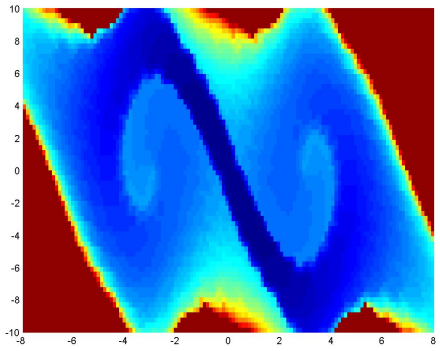

Abbildung 7.21: $V_{\mathcal{P}}$ mit 8192 Zellen

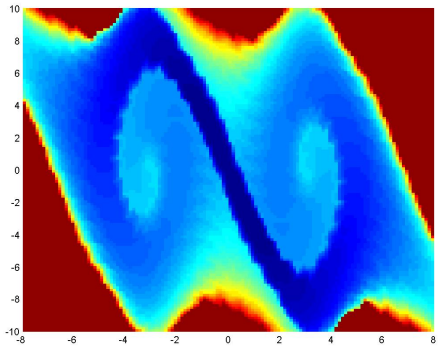

Abbildung 7.22: $V_{\mathcal{P}}$ mit 16384 Zellen

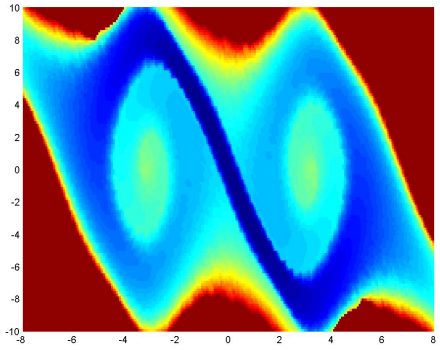

Abbildung 7.23: $V_{\mathcal{P}}$ mit 32768 Zellen

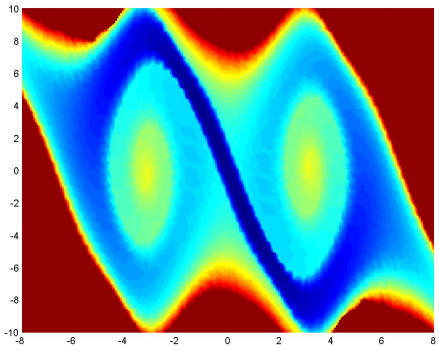

Abbildung 7.24: $V_{\mathcal{P}}$ mit 65536 Zellen

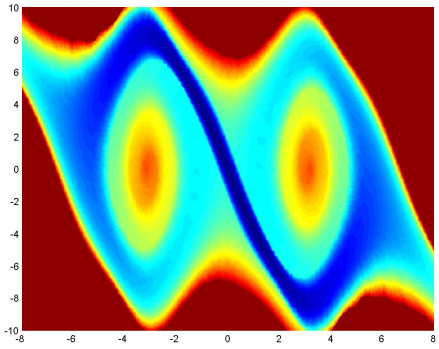

Abbildung 7.25: $V_{\mathcal{P}}$ mit 131072 Zellen

Wendet man dagegen die adaptive Verfeinerung mit $\theta = 0.8$ an, so unterscheiden sich beide Verfeinerungen anfangs nicht, erst ab der achten Verfeinerung verhält sich die adaptive anders als die gleichmäßige. Man erhält folgende Egebnisse:

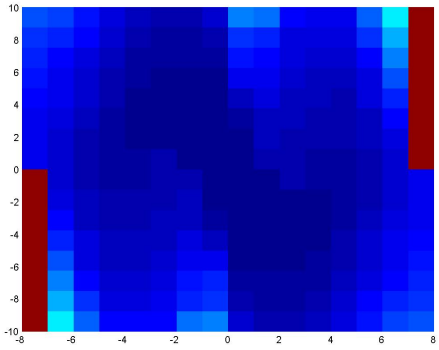

Abbildung 7.26: $V_{\mathcal{P}}$ mit 251 Zellen

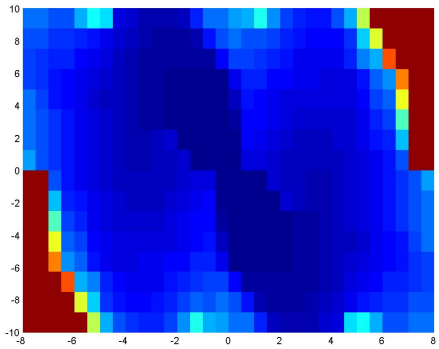

Abbildung 7.27: $V_{\mathcal{P}}$ mit 497 Zellen

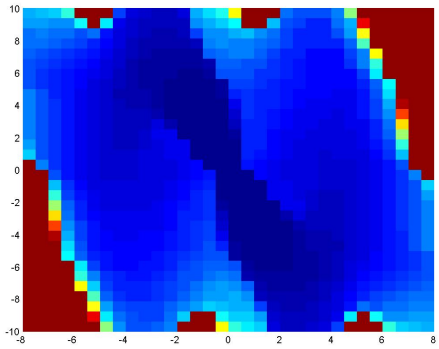

Abbildung 7.28: $V_{\mathcal{P}}$ mit 950 Zellen

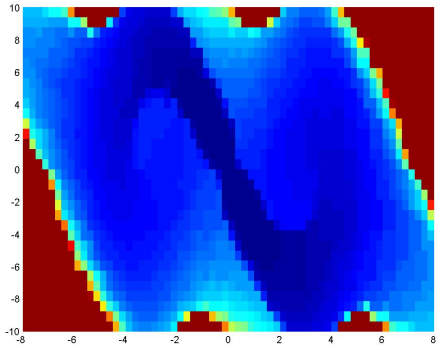

Abbildung 7.29: $V_{\mathcal{P}}$ mit 1798 Zellen

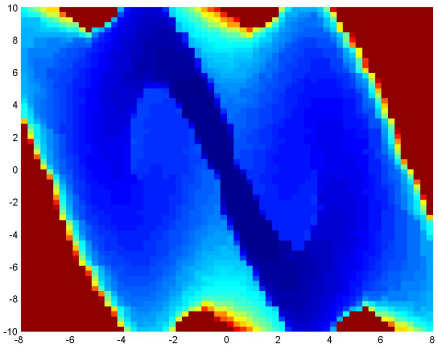

Abbildung 7.30: $V_{\mathcal{P}}$ mit 3324 Zellen

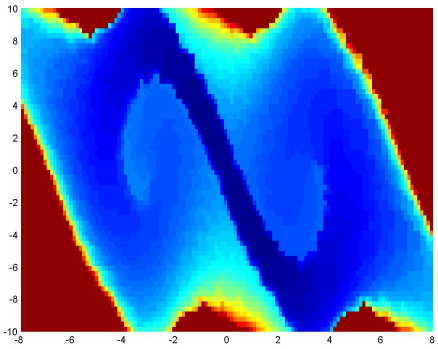

Abbildung 7.31: $V_{\mathcal{P}}$ mit 6046 Zellen

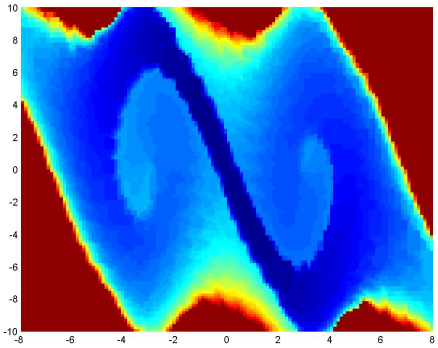

Abbildung 7.32: $V_{\mathcal{P}}$ mit 10828 Zellen

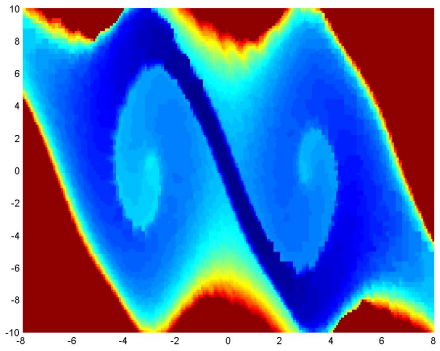

Abbildung 7.33: $V_{\mathcal{P}}$ mit 18912 Zellen

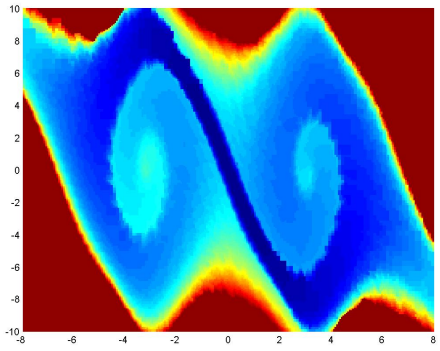

Abbildung 7.34: $V_{\mathcal{P}}$ mit 32479 Zellen

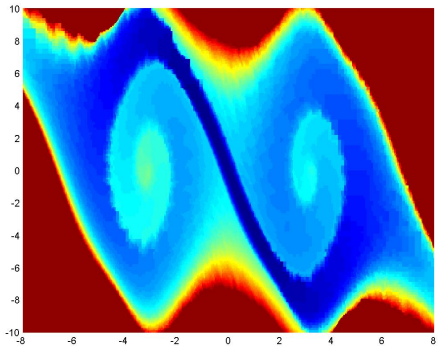

Abbildung 7.35: $V_{\mathcal{P}}$ mit 55161 Zellen

Die Stabilität der Feedback-Kontrolle kann erst bei sehr feiner Zellenüberdeckung beobachtet werden. Es folgen drei Bilder, die die Trajektorie mit Startwert $(0.1, 0.1)$ betrachten, das erste

Bild bei einer Überdeckung mit 4096 Zellen, das zweite Bild bei einer Übereckung mit 32768 Zellen und das dritte bei 131072 Zellen. Alle drei Übedeckungen sind gleichmäßig.

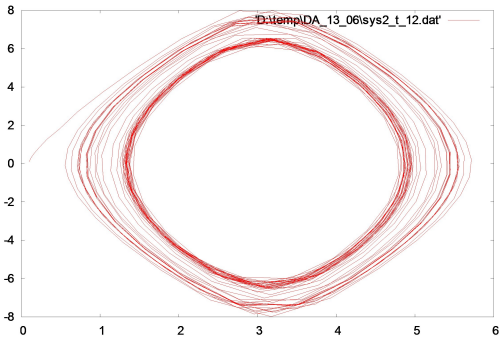

Abbildung 7.36: Trajektorie bei 4096 Zellen

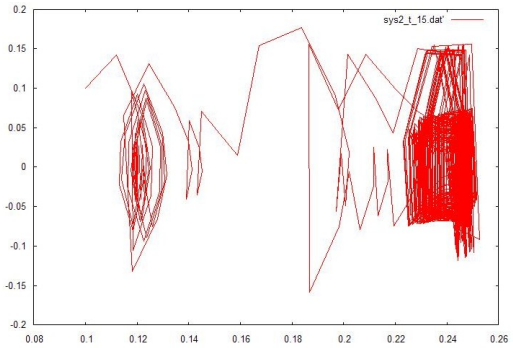

Abbildung 7.37: Trajektorie bei 32768 Zellen

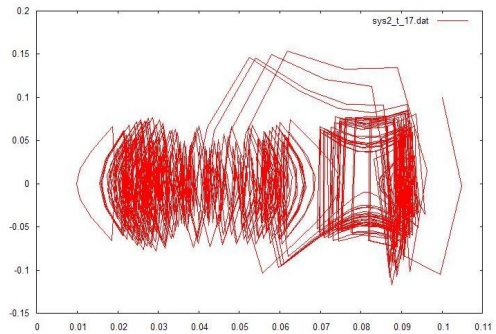

Abbildung 7.38: Trajektorie bei 131072 Zellen

Besonders im ersten Bild ist die Stabilität nicht gegeben.

Nun wird die Trajektorie mit Startwert $(3.1, 0.1)$ bei den gleichen Zellenüberdeckungen betrachtet.

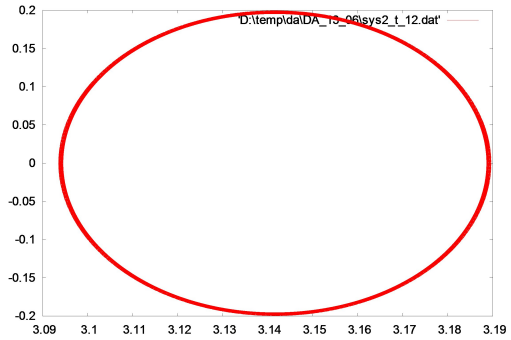

Abbildung 7.39: Trajektorie bei 4096 Zellen

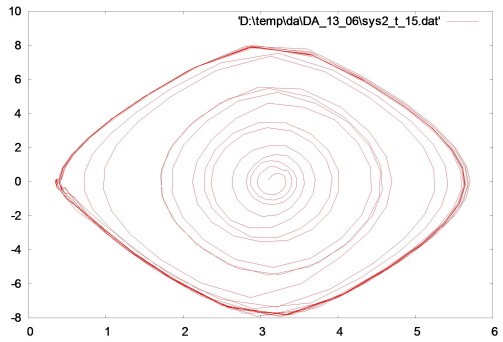

Abbildung 7.40: Trajektorie bei 32768 Zellen

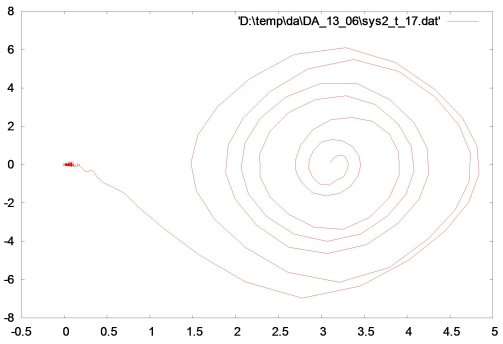

Abbildung 7.41: Trajektorie bei 131072 Zellen

Nur im dritten Bild ist die Stabilität der Feedback-Kontrolle gegeben.

Bei der adaptiven Verfeinerung mit 55161 Zellen ist die Stabilitat der Feedbackkontrolle nicht gegeben, wie folgende Bilder zeigen.

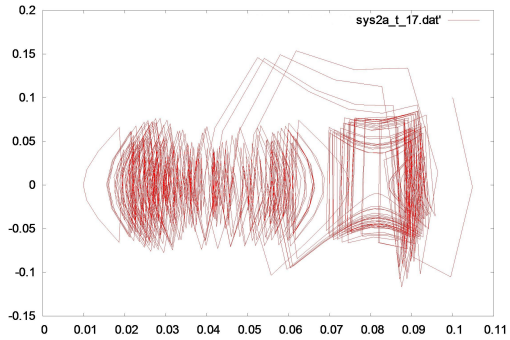

Abbildung 7.42: Trajektorie mit Startwert $(0.1, 0.1)$

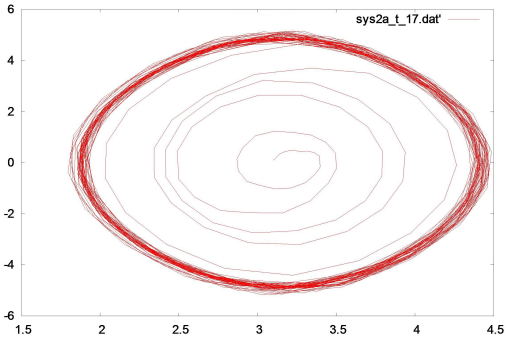

Abbildung 7.43: Trajektorie mit Startwert $(3.1, 0.1)$

Kapitel 8

Ausblick

Wie kann die Effizienz dieses graphentheoretischen Ansatzes weiter gesteigert werden?

Ein Ansatz liegt in einer verbesserten Verfeinerungsstrategie. In meiner Implementierung werden nach einer Verfeinerung des Gitters Daten, die unverändert übernommen werden können, nicht neu berechnet. Also sollte sich durch eine verbesserte Verfeinerungsstrategie der Rechenaufwand reduzieren lassen.

Ein Nachteil dieses graphentheoretischen Ansatzes besteht darin, dass die Wertefunktion auf einer Zelle als konstant approximiert wird. Es ist zu überlegen, ob man mittels dieses Ansatzes bzw. eines modifizierten Ansatzes eine nicht-konstante Approximation auf den einzelnen Zellen erreichen kann. Eine auf den einzelnen Zellen nicht-konstante Approximation lässt auf eine bessere Approximation hoffen.

In [9] wird der graphentheoretische Ansatz auf gestörte Systeme der Form

$$x_{k+1} = f(x_k, u_k, w_k), \quad k = 0, 1, \ldots \tag{8.1}$$

erweitert, wobei $f : X \times U \times W \longrightarrow \mathbb{R}^d$ eine stetige Funktion und $X \subset \mathbb{R}^d$, der Kontrollwertbereich $U \subset \mathbb{R}^m$ und der Raum der Störwerte $W \subset \mathbb{R}^l$ ist. In dieser Arbeit wird mit Hilfe einer spieltheoretischen Methode die Berechnung der Wertefunktion auf ein Kürzestes-Wege-Problem auf einem Hypergraphen zurückgeführt. Dieses Problem wird mit einer modifizierten Version des Dijkstra-Algorithmus gelöst.

Des Weiteren ist es einer Überlegung wert, ob und wie dieser Ansatz auf Kontrollsysteme, die durch partielle Differentialgleichungen gegeben sind, anzuwenden ist.

Anhang A

Inhalt der CD

Auf der beiliegenden CD sind folgende Dateien enthalten:

`da.pdf`	Diese Arbeit im PDF-Format
`Makefile`	Makefile zum Kompilieren der Programme
`defs.h`	Header-Datei mit grundlegenden Definitionen, die von der Gitterverwaltung verwendet werden
`grid.h`	Header-Datei für die Gitterverwaltung
`grid.c`	Implementierung der Gitterverwaltung
`rk_eingb.h`	Header-Datei für den Differentialgleichungslöser
`rk_eingb.c`	Implementierung des Differentialgleichungslösers
`system1.c`	Beispielsystem 1 mit gleichmäßiger Gitterverfeinerung
`system1a.c`	Beispielsystem 1 mit adaptiver Gitterverfeinerung
`system2.c`	Beispielsystem 2 mit gleichmäßiger Gitterverfeinerung
`system2a.c`	Beispielsystem 2 mit adaptiver Gitterverfeinerung

In `system1.c`, `system1a.c`, `system2.c` und `system2a.c` wird jeweils eine Folge von Zellenüberdeckungen betrachtet, die optimale Wertefunktion und eine Trajektorie berechnet und in eine Datei ausgegeben.

Anhang B

Notation

In der folgenden Übersicht sind einige wichtige Bezeichnungen und Notationen vermerkt, die im Laufe der vorliegenden Arbeit regelmäßig erscheinen.

BEZEICHNUNG	BEDEUTUNG
$\|E\|$, wobei E eine Menge ist	Mächtigkeit von E
$\|x\|, x = (x_1\ x_2\ \ldots\ x_n)^T \in \mathbb{R}^n$	euklidische Norm: $\|x\| = \sqrt{\sum_{i=1}^{n} x_i^2}$
$\text{int}D, D \subset \mathbb{R}^n$	das Innere von D
$\overline{D}, D \subset \mathbb{R}^n$	der Abschluss von D
$\text{diam}(D), D \subset X, (X, d)$ metrischer Raum	$\text{diam}(D) := \sup_{x,y \in D} d(x, y)$ ist der Durchmesser von D
Landau-Symbol O: $f \in O(g)$	$\exists\, c \in \mathbb{R}$ mit $f(n) \leq g(n)$ für fast alle $n \in \mathbb{N}$

Literaturverzeichnis

[1] D. Bertsekas, *Dynamic Programming and Optimal Control* (2. Auflage); Athena Scientific, Belmont, 2000.

[2] J. Clark, D. Holton, *Graphentheorie*; Spektrum Akademischer Verlag, Heidelberg, 1994.

[3] M. Dellnitz, A. Hohmann, *A subdivision algorithm for the computation of unstable manifolds and global attractors*; Numerische Mathematik 75 (3) (1997), 293-317.

[4] E. Dijkstra, *A note on two problems in connexion with graphs*; Numerische Mathematik 1 (1959), 269-271.

[5] L. Grüne, *Numerische Mathematik II: Differentialgleichungen*; Vorlesungsskript; `http://www.uni-bayreuth.de/departments/math/~lgruene/numerik03/`.

[6] L. Grüne, *Numerik Dynamischer Systeme*; Vorlesungsskript; `http://www.uni-bayreuth.de/departments/math/~lgruene/numdyn0304/`.

[7] L. Grüne, *Numerische Dynamik von Kontrollsystemen*; Vorlesungsskript; `http://www.uni-bayreuth.de/departments/math/~lgruene/ndks04/`.

[8] L. Grüne, O. Junge, *A set oriented approach to optimal feedback stabilization*; Systems & Control Letters, 54 (2) (2005), 169-180.

[9] L. Grüne, O. Junge, *Global optimal control of perturbed systems*; noch nicht veröffentlicht; `http://www.uni-bayreuth.de/departments/math/~lgruene/publ/robsetstab.html`.

[10] O. Junge, H. Osinga, *A set oriented approach to global optimal control*; ESAIM: Control, Optimisation and Calculus of Variations 10 (2) (2004), 259-270.

[11] H. Khalil, *Nonlinear Systems* (2. Auflage); Prentice Hall, Upper Saddle River, 1996.

[12] D. Nesic, A. Teel, *A framework for stabilization of nonlinear sampled-data systems based on their approximate discrete-time models*, IEEE Trans. Autom. Control 49 (2004), 1103-1122.

[13] T. Ottmann, P. Widmayer, *Algorithmen und Datenstrukturen* (3. Auflage); Spektrum Akademischer Verlag, Heidelberg, 1996.

[14] A. Schrijver, *Combinatorial Optimization: Polyhedra and Efficiency, Vol. A*; Springer Verlag, Heidelberg, 2003.

[15] E. Sontag, *Mathematical Control Theory: Deterministic Finite Dimensional Systems* (2. Auflage); Springer Verlag, New York, 1998.

ERKLÄRUNG

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen.

Hof, den 16. November 2005 .....

Marcus von Lossow